INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700  800/521-0600
Knowledge-based information systems development with an organizational perspective

Mangal, Ram Kishore, Ph.D.

The Ohio State University, 1993
KNOWLEDGE-BASED INFORMATION SYSTEMS DEVELOPMENT WITH AN
ORGANIZATIONAL PERSPECTIVE

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
The Degree Doctor of Philosophy in the Graduate
School of The Ohio State University

By
Ram Kishore Mangal, B.Tech., P.G.D.M.

*****

The Ohio State University
1993

Dissertation Committee:
H. Pirkul
V.S. Jacob
W.A. Muhanna

Approved by

H. Pirkul (Adviser)
Department of Accounting
And M.I.S.
ACKNOWLEDGEMENTS

I express sincere appreciation to Dr. Hasan Pirkul for his guidance and insight throughout the research. Thanks go to other members of my advisory committee, Drs. Vargheese Jacob and Waleed Muhanna, for their suggestions and comments:
VITA

April 15, 1957 ............... Born - Ajmer, India

1978 ......................... B.Tech., Indian Institute of Technology, Kanpur, India

1980 ......................... P.G.D.M., Indian Institute of Management, Ahmedabad, India

1980-1985 ..................... Sr. Officer, M.I.S., Grindwell Norton limited, Bombay, India

1985-1989 ..................... Manager E.D.P., Housing Development Finance Corporation, Bombay, India

1993-Present .................. Assistant Professor, School of Accountancy, Memphis State University, Memphis, Tennessee

FIELDS OF STUDY

Major field of study: Accounting and Management Information Systems
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS ........................................ ii
VITA ....................................................... iii
LIST OF FIGURES .......................................... vi

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. INTRODUCTION .........................</td>
<td>1</td>
</tr>
<tr>
<td>II. LITERATURE REVIEW ..................</td>
<td>7</td>
</tr>
<tr>
<td>1. Systems Development Overview ........</td>
<td>7</td>
</tr>
<tr>
<td>2. Need for Automated Support ..........</td>
<td>10</td>
</tr>
<tr>
<td>3. CASE Tools for Supporting Systems Development</td>
<td>13</td>
</tr>
<tr>
<td>4. Knowledge based approaches ..........</td>
<td>15</td>
</tr>
<tr>
<td>5. Organizational Modeling and Organization Specific IS Development Support</td>
<td>20</td>
</tr>
<tr>
<td>6. Summary ....................................</td>
<td>22</td>
</tr>
<tr>
<td>III. THE ORGANIZATIONAL SYSTEMS DEVELOPMENT MODEL .</td>
<td>24</td>
</tr>
<tr>
<td>1. Introduction ..............................</td>
<td>24</td>
</tr>
<tr>
<td>2. The OSD model ............................</td>
<td>25</td>
</tr>
<tr>
<td>3. The OSD process .........................</td>
<td>31</td>
</tr>
<tr>
<td>3.1 Specification development process ...</td>
<td>31</td>
</tr>
<tr>
<td>3.2 System Synthesizer Process ..........</td>
<td>32</td>
</tr>
<tr>
<td>4. Organizational Systems Development Environment ..................</td>
<td>35</td>
</tr>
<tr>
<td>5. Organizational Knowledge ..............</td>
<td>36</td>
</tr>
<tr>
<td>5.1 Requirements and the Role of Organizational Knowledge ..........</td>
<td>36</td>
</tr>
<tr>
<td>5.2 Organization knowledge as a collection of knowledge ..........</td>
<td>39</td>
</tr>
<tr>
<td>6. IS knowledge and Organization Model ......</td>
<td>42</td>
</tr>
<tr>
<td>7. Systems development in the OSD model - an overview ........</td>
<td>45</td>
</tr>
<tr>
<td>8. Summary ....................................</td>
<td>48</td>
</tr>
<tr>
<td>IV. ORGANIZATION MODEL and IS DEVELOPMENT ................</td>
<td>50</td>
</tr>
<tr>
<td>1. Introduction ..............................</td>
<td>50</td>
</tr>
<tr>
<td>2. Representation of OM ...................</td>
<td>50</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>2.1 Evaluation criteria</td>
<td>51</td>
</tr>
<tr>
<td>2.2 Comparison of some common representations</td>
<td>54</td>
</tr>
<tr>
<td>3. Object-Oriented Representation for OM</td>
<td>56</td>
</tr>
<tr>
<td>3.1 Object-oriented meta-model for OM</td>
<td>59</td>
</tr>
<tr>
<td>3.2 Object categories in OM</td>
<td>61</td>
</tr>
<tr>
<td>4. View and OM Related Algorithms</td>
<td>63</td>
</tr>
<tr>
<td>5. OM-based IS development process</td>
<td>67</td>
</tr>
<tr>
<td>6. Advantages of OM-based IS development</td>
<td>71</td>
</tr>
<tr>
<td>7. Summary</td>
<td>74</td>
</tr>
<tr>
<td>V. ARCHITECTURE OF THE OSD MODEL</td>
<td>76</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>76</td>
</tr>
<tr>
<td>2. Architecture for the OSD model</td>
<td>77</td>
</tr>
<tr>
<td>3. Specifier</td>
<td>81</td>
</tr>
<tr>
<td>4. Organizational Knowledge Base</td>
<td>87</td>
</tr>
<tr>
<td>5. Domain Knowledge Base</td>
<td>89</td>
</tr>
<tr>
<td>6. Systems Development Knowledge Base</td>
<td>90</td>
</tr>
<tr>
<td>7. IS Knowledge Base (ISKB)</td>
<td>92</td>
</tr>
<tr>
<td>8. Summary</td>
<td>93</td>
</tr>
<tr>
<td>VI. SPECIFIER</td>
<td>94</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>94</td>
</tr>
<tr>
<td>2. The Organization Model Structure</td>
<td>95</td>
</tr>
<tr>
<td>3. Knowledge as Rules and Assertions</td>
<td>100</td>
</tr>
<tr>
<td>4. Reasoning Mechanisms</td>
<td>102</td>
</tr>
<tr>
<td>5. View Creation, Generation, Management, and Validation</td>
<td>103</td>
</tr>
<tr>
<td>6. User Interface and Interaction with the Specifier</td>
<td>105</td>
</tr>
<tr>
<td>7. Conclusions</td>
<td>122</td>
</tr>
<tr>
<td>VII. SUMMARY AND FUTURE RESEARCH DIRECTIONS</td>
<td>124</td>
</tr>
<tr>
<td>APPENDICES</td>
<td></td>
</tr>
<tr>
<td>A. FRAMES IN THE SPECIFIER</td>
<td>129</td>
</tr>
<tr>
<td>B. HANDLERS IN THE SPECIFIER</td>
<td>134</td>
</tr>
<tr>
<td>C. RULES IN THE SPECIFIER</td>
<td>138</td>
</tr>
<tr>
<td>LIST OF REFERENCES</td>
<td>152</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<p>| FIGURES |
|--------------------------|----------|
| 1. Systems Development Life Cycle | 7 |
| 2. The OSD Model | 26 |
| 3. OM-based application development model | 46 |
| 4. Object representation scheme for the OM | 58 |
| 5. Organization Model Meta-model | 60 |
| 6. Sales-report and generic objects | 62 |
| 7. Objects creation in OM-based IS development | 69 |
| 8. System development processes in OM-based IS development | 70 |
| 9. Reuse-oriented OM-based IS development model | 72 |
| 10. Architecture for the OSD model | 80 |
| 11. Architecture for the Specifier | 85 |
| 12. Lattice of OM related frames in Specifier | 96 |
| 13. Instance of an Application frame | 99 |
| 14. View - pending issues | 106 |
| 15. Organizational Model pending issues | 107 |
| 16. Specifier screen with menu bar | 108 |
| 17. Pick-list of existing applications in the OM | 109 |
| 18. Application object ENQUIRY | 111 |
| 19. ENQ-QUO the application view | 112 |
| 20. OM-OBJECT: ENQUIRY window | 113 |</p>
<table>
<thead>
<tr>
<th>Number</th>
<th>Specification</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Specifier: Options for Org-model</td>
<td>115</td>
</tr>
<tr>
<td>22</td>
<td>Organization model object CUSTOMER</td>
<td>116</td>
</tr>
<tr>
<td>23</td>
<td>Specifier: View options</td>
<td>117</td>
</tr>
<tr>
<td>24</td>
<td>Specifier: Enhance-view options</td>
<td>118</td>
</tr>
<tr>
<td>25</td>
<td>Specifier: Enhance model screens</td>
<td>119</td>
</tr>
<tr>
<td>26</td>
<td>Specifier: Select objects from organization model window</td>
<td>119</td>
</tr>
<tr>
<td>27</td>
<td>Specifier: Reuse analysis for a software component type</td>
<td>120</td>
</tr>
<tr>
<td>28</td>
<td>Specifier: Validate options</td>
<td>121</td>
</tr>
</tbody>
</table>
Organizations today are increasingly relying on information technologies for a vast array of activities including restructuring business processes, achieving fast-cycle capability for competitive power, and forming strategic alliances of information systems with other organizations. As a result, information systems are becoming increasingly complex, and organization-wide information systems that integrate business functions and support new business opportunities are becoming imperative [Sawy89, Bowe88, Hamm90, Bowe90, Quin90]. A recent study shows that boosting systems development productivity, instituting cross-functional systems and integrating information systems (IS) are some key issues IS professionals need to address today [Carr91]. Our research focuses on a systems development (SD) environment that addresses and provides solutions to these issues.

It is widely recognized that system development as well as system maintenance are complex, labor and knowledge intensive activities, and are error prone. Correct and timely incorporation of modifications and enhancements in large-scale interconnected information systems are complex,
nontrivial tasks [Zmud84]. Information systems that provide organization-wide functionality and are vertically integrated require substantial knowledge specific to the organization. Also, design knowledge of existing information systems is required for proper integration and for effecting consistent modifications across multiple information systems. Thus, system development in an organization can be facilitated by providing systematic support of these knowledge.

Many knowledge-based and software engineering approaches have emerged to improve the effectiveness and efficiency of systems development (SD). Emphasis varies from assisting system developers particularly during analysis and design phases, to automatic programming. A review of current knowledge-based approaches for supporting SD shows that they have typically used domain knowledge and system development knowledge to provide intelligent support. These approaches, in general, do not provide knowledge that is specific to the organization and knowledge about existing information systems within the organization for software development. This lack an organizational perspective hampers integration and effecting consistent modification across multiple information systems.

Current knowledge-based approaches also do not adequately reuse related analysis and design information available in the organization. They do not satisfactorily address the issues related to the development of vertically
integrated systems like integration of transaction processing and decision support systems. Moreover, they do not adequately focus on adaptability and changeability of system [Sawy89] nor do they provide mechanisms to accomplish quick, easy and correct modifications. Clearly, a substantial gap exists between the support required for SD within an organization today, and the support provided by the current approaches.

We propose an integrated model, the Organizational Systems Development (OSD) model, which provides intelligent support to meet the challenges of IS development, integration and maintenance in an organizational environment. The OSD model addresses integration and adaptability issues by integrating the system specifications of all information systems in the organization into one organization model (OM). Specific system specifications are then considered views of this model. We call the system development using the organization model as OM-based system development.

The OSD model uses organizational knowledge, knowledge about other systems in the organization, and generic knowledge about different domains to facilitate development; and system development knowledge to validate the system being developed and to generate code. We refer to the resulting systems development environment in the OSD model as the organizational systems development environment (OSDE). An object-oriented representation scheme is adopted to manage
complexity, maximize reuse, enhance productivity and provide robustness to IS. The OSD model supports effecting correct, consistent, and speedy modifications in IS.

The OSD model has two main processes, the specifications development process, and the system synthesizer process. The specifications development process develops the specifications; the system synthesizer process transforms these specifications into code. In this dissertation, we focus primarily on the specifications development process.

Use of the OSD model will facilitate complete IS development cycle - from accepting user requirements, to generating an implementable system, and maintaining it. OSD is intended to support all types of systems, such as transaction processing, decision support, organization support, strategic information systems, inter-organizational systems, or any combination of these.

This research makes a significant contribution to the systems development domain. Unlike prior approaches, we propose a unified approach for systematically using various types of knowledge, including organizational knowledge and knowledge about other systems in the organization. Some contributions of our research presented here are as follows.

1. We introduce and discuss the concepts of the organization model, application views, and organization-model based systems development.
2. We provide framework to use knowledge of the organization during system development to make the system suitable for the environment in which it will operate.

3. We develop criteria for evaluating representations for the specifications from the point of reuse, knowledge content in the application model, and maintainability.

4. We present the OM-based systems development that
   (a) considers other systems the system being developed will interact with;
   (b) reuses past analysis and design information available in the organization to improve productivity and to avoid inconsistencies;
   (c) links the specifications with the business policies;
   (d) enables quick and correct maintenance when a single change affects multiple systems in the organization;
   (e) inherently supports reuse and integration;
   (f) provides information system about various systems in the organization;

5. We classify and identify the roles of domain, organizational, and system development knowledge during systems development.

6. We present the OSD environment that provides necessary organizational perspective during systems development and organizes knowledge used such that exchange of
knowledge between different OSD environments is possible.

The rest of the dissertation is organized as follows. Chapter II reviews related work. Chapter III discusses the OSD model and its components, the OSD environment model, the roles of various knowledge bases and the rationale for including them in the OSDE. Since the organization model is most critical component of the OSD model, we discuss it in detail in chapter IV and illustrate how it supports inherent integration, reuse, and modifications in the OM-based systems development. Chapter IV also develops criteria for evaluating the representation for the organization model, and compare some common representation schemes with our object-oriented representation. Chapter V discusses an architecture for implementing the OSD model. Chapter VI discusses the Specifier, our current implementation of the specification development process that demonstrates the key features of OM-based systems development. Our conclusions and direction for future research are presented in chapter VII.
CHAPTER II
LITERATURE REVIEW

1. Systems Development Overview

Systems development process gets initiated as a result of some business need. Output of this development process is an information system that business uses. Any change in the organization or in its need for information is fed back to the systems development process, and the existing system is modified or a new system is developed. Figure 1 briefly describes this systems development life cycle [Kroe92].

![Figure 1: Systems Development Life Cycle](image)

Business ----> Systems --------> Information -------> Business needs
| Development System | Change Process |
|----------<--------------------------T--------------< -

Systems development process broadly consists of three phases: requirements analysis and modeling, design, and implementation [Chun91][Jalo91]. Requirements analysis is the process of determining and documenting the customer's needs and constraints. This stage involves transferring the requirements and knowledge of a problem domain from the
user(s) to the analyst who develops formal specifications and models the processes involved. Requirements specifications developed during the requirements analysis stage are used during the design stage for developing design specifications. During implementation, design specifications are converted into an implementable code and tested.

A software development process model specifies how the above phases and associated activities are organized in the entire software development effort. Some of the well-known models are the waterfall model [Somm88], the iterative enhancement model [Basi75], the spiral model [Boeh88], and the transformation model [Ghez91].

The waterfall model is criticized as rigid, pre-specified and non-adaptive [Ghez91]. The common features of other models is their lack of linearity with respect to the development phases.

System development methodologies provide a systematic and disciplined approach for developing systems. Structured development methodology is a popular methodology used for systems analysis and design. It uses top-down method to define system requirements and to design. Structured analysis [Gane77] [Marc78] employs a top-down technique in which the first step involves the description of high-level flows, then successive refinement of detailed data flows. The system
specification produced is composed of data flow diagrams\(^1\) (DFD), a data dictionary, and process specifications.

Jackson Structured Development (JSD) [Jack83] is another methodology based on top-down approach. It is popular in Europe. A JSD model describes the real world in terms of entities, actions and ordering of actions. Specifications are represented using graphic models and pseudo codes. JSD has been criticized as ineffective at abstraction and simplification, and as an inadequate methodology for database systems [Rumb91].

Formal specification methods employ mathematical notation to model the system. They support the production of a complete, unambiguous, non contradictory requirements specification. Vienna Development Method (VDM) [Jone86] and Z [Pott91] are examples of formal specification methods. These approaches are not easy for beginners and are not a viable communication interface with the users. They lack structure and are therefore not suitable for large scale development. Their use is generally preceded by the use of an informal methodology. Semmens et al suggest some integrated informal and formal specification techniques [Semm92].

Object oriented methodologies are slowly gaining popularity. An object-oriented methodology models a situation

---

\(^1\)A data-flow diagram is a network representation of a system showing the system processes and the data connecting those processes.
in terms of collection of interacting entities, each of which
encapsulate data and related actions. They differ from data
oriented methods which regard data and related actions as
separate entities. Object oriented analysis (OOA) and design
(OOD) emphasize and are based on the object model, which
closely corresponds to the real world [Booc88]. This is in
contrast to structured analysis and design which emphasizes
the functional model. Advantages include homogeneity of
representation during analysis and design, extensibility of
the model and specifications, and relative stability of the
model compared to DFD based specifications [Rumb91].

2. Need for Automated Support

In the late 1950s and early 1960s, programs written were
generally small, for a well defined task and for use by one
person. When truly large software systems were commercially
attempted during the middle to late 60s, numerous problems
were encountered. It was realized that problems were not well
understood. Also, techniques of small programming cannot be
scaled up for large software projects. Large software
projects were generally over-budget and behind schedule, and
did not meet user needs. The term "software crisis" was
invented during this period to refer to these software
problems.

Realizing that software development and management needs
a disciplined engineering approach, numerous methodologies to
overcome software development problems were proposed. Focus of the research was apparently on proposing new better methods, improving upon the existing software development process, and on representation of the intermediate software artifacts like specifications.

In spite of using software engineering methodologies, software systems continued to have problems. Studies show that many problems occur due to human error during software development [Youn74]. Nakajo analyzed the software error cause-effect relationships when using structured methodologies and showed that "the structured methods could suppress the error-occurrence mechanisms, being more effective on hardware and software interface mechanisms than on system and module function mechanisms" [Naka91]. Brooks argued that problems are due to inherent nature of the software development process itself [Broo87]. Arguing that software is fundamentally more complex than other artifacts, he pointed out that (a) software systems have fewer repeating elements, (b) they undergo repeated modifications, and (c) scaling up software systems have non-linear effect on complexity.

As scaling-up has non-linear effect on complexity, problems associated with developing large systems are much more severe. Curtis et al studied the problems of designing large software systems and analyzed how "..the thin spread of application domain knowledge, fluctuating and conflicting
requirements, and communication bottlenecks and breakdowns - affected software productivity and quality through their impact on cognitive, social, and organizational processes" [Curt88]. Also, maintaining such large systems is much more complex and nontrivial [Zmud84], resulting in high backlog and high cost. Maintenance costs are reported to be as high as 80% for large systems.

Various studies like above has led to the conclusion that systems development is a complex, labor and knowledge intensive activity. It needs to be supported. Need for support is increasing with the complexity of information systems and rapidly changing environments in which organizations operate today.

Organizations are moving towards increased inter-organizational information sharing and trans-organizational alliances. They are increasingly relying on information technologies for, among others, restructuring business processes and achieving fast-cycle capability for competitive power. A recent study shows that boosting systems development productivity, instituting cross-functional systems and integrating information systems (IS) are some key issues IS professionals face today [Carr91].

Complex information systems that provide organization-wide functionality, and are vertically integrated, require substantial knowledge specific to the organization. Also, design knowledge of existing information
systems is required for proper integration, and for effecting consistent modifications across multiple information systems. Thus need for proper support in today's environment for information systems (IS) development is immense.

Many approaches have emerged to support and to improve the effectiveness and efficiency of systems development (SD). Emphasis varies from assisting system developers particularly during analysis and design phases, to automatic programming. Support approaches can be broadly classified into conventional approaches and knowledge-based approaches. Prominent among conventional approaches are computer-aided software engineering (CASE) tools. Knowledge-based approaches use knowledge and reasoning mechanisms to make system development more effective, with the goal of automating the process as much as possible. Research in this area is generally classified as knowledge-based software engineering (KBSE). In the following sections, we discuss the support mechanisms in these two areas.

3. CASE tools for supporting systems development

The term CASE was used first in 1984 to focus attention on tools which automated structured analysis and design [Shar90]. It has a much wider connotation today. The primary goal of CASE tools is to automate the most error-prone and labor-intensive tasks in the software development process. Other goals of CASE environments are to speed up software
development and to enable construction of higher-quality software [Berz91].

CASE tools provide support by performing many conceptually similar tasks. They boost productivity and prevent representation related defects in the software development process by automating routine software development tasks, removing clerical overheads, and enforcing rigorous design rule checking [Fort92]. CASE products support one of many software development methodologies. Most of the available CASE tools support systems development based on structured methodologies. CASE tools supporting object-oriented analysis and design are now becoming commercially available [Coad91].

Although CASE tools have many advantages, they have their limitations. Their focus is primarily towards improving the efficiency of the software development process. CASE design tools help users in representing the designs correctly (i.e., support the design process); they do not help users in ensuring that the correct product is being designed. Also, CASE tools do not record design knowledge that led to a particular design; this knowledge is useful for maintaining the system. Moreover, most of the CASE tools do not provide support for the complete SD cycle [Fort92]. Of the seven support aspects during design identified by Lubars\(^2\) [Luba89],

\(^2\)Lubars identified the following seven support aspects: clerical, interface, analysis, testing, organizational, knowledge-based, and intelligent support [Luba89].
CASE tools generally provide clerical, interface, design evaluation, and may be testing support.

4. Knowledge based approaches

Knowledge-based approaches use knowledge and reasoning mechanisms to make systems development more effective. A knowledge-base is a database augmented with rules for reasoning about the data and an inference engine that applies the rules. This reasoning capability is used to reason about the semantics of a system under development — a significant improvement over approaches merely reasoning about the structure of a system. Systems have been suggested to provide intelligent assistance at different stages of the software life cycle — from specification acquisition to program synthesis.

Several knowledge-based approaches for supporting the requirements analysis stage have been proposed. During this stage, user requirements (what needs to be done) are analyzed to develop the specifications (what will the system do) for the system. Research has shown that a large number of errors can be traced to this stage [Curt88][Miri91]. This is a crucial phase as the system being developed and implemented is based on it. Thus, mistakes at this stage are obviously very costly. Many intelligent approaches for assisting the design stage have also been proposed. These approaches provide assistance in designing from specifications, and in
evaluating and refining designs; and assume that requirements specifications are available. Some approach address both stages. We briefly review some of the approaches.

The Requirements Apprentice (RA) project at MIT focuses on the formalization of requirements [Reub91]. It provides assistance in removing ambiguity, contradiction, and incompleteness from the requirements by using dependency directed reasoning mechanisms and knowledge about commonly occurring structures (clichés) in the domain. Clichés are stored in the cliché library, a declarative repository, and provide bulk of general information about the domain. They are represented as frames linked by constraints and arranged in an inheritance hierarchy. Clichés support the developer in framing the problem. The RA approach, however, is based on the waterfall model. It does not allow designers to interweave analysis and design.

Johnson [John88] describes a transformational approach for incremental construction of valid formal specifications from descriptions and provides intelligent support for applying evolution transformations. The ARIES prototype supports requirements acquisition and analysis. Its approach is centered "around selection of appropriate evolution transformations, and reformulating abstract descriptions of system behavior using such transformations." [John92]. Gist is used as specification language [John88]. Gist consist of definitions of types, relations, procedures, and constraints.
Domain knowledge is organized as folders, which are collections of reusable concept definitions. Concepts are represented as semantic nets [John92]. Aries supports the analyst, not the designer.

Fickas and Nagarajan provide intelligent support for critiquing and modifying formal specifications using domain specific knowledge [Fick88]. Critic, part of the Kate project, critiques by matching the specification with parts of the domain model. Domain knowledge is organized into specification goals and cases. A case is described using four fields - operation description, links to goals, scenarios, and descriptions. The process is interactive and user is required to assist in bindings whenever critic is not able to resolve the ambiguity.

Draco also uses domain knowledge for supporting systems development. It stores expert knowledge of domain analysts and domain designers in the knowledge base and makes them available for reuse as software components [Neig84]. The domain knowledge is made available for use in terms of domain concepts which enables easy reuse and better communication with the users. Multiple design components are available for the same specification for system designer to choose from. Draco approach thus supports reuse of analysis and design as compared to reusing of code. Draco however does not reuse the specifications and designs to help developers frame new problems.
ITHACA applications development environment supports IS development by stocking a software base with reusable components for various domains represented in object-oriented terms [Fugi92]. Its approach is similar to ours to the extent that it uses object-oriented techniques at all levels of development. ITHACA has a set of application tools that supports developer in finding and combining various software components into complete application.

The above approaches use domain knowledge (e.g., knowledge about a typical inventory system) to improve productivity and to improve the quality of the requirements specifications. However, they do not ensure proper integration with other IS in the organization; they also do not use knowledge about the organization to ensure that the specifications are consistent with organization's procedures and policies. A major drawback of the above approaches is that they consider requirements analysis in an environment isolated from that of the organization.

Various approaches have been suggested to support the design phase. Miriyala and Harandi derive formal specifications of data types and programs from informal descriptions using schemas, analogy and difference-based reasoning [Miri91]. Allen and Lee present a model for software parts composition process and provide a knowledge-base user interface to Ada software libraries [Alle89]. The Intelligent Design Aid (IDeA) performs top-down refinement of
specifications into design, using design schemas and design 
rules for refinement and specialization [Luba87, Luba89, 
Luba86]. IDeA represents specifications and designs as data 
flow diagrams (DFDs). Tsai and Ridge, in their Specification 
Transformation Expert System (STEP), also automatically 
translate specifications into design [Tsai88]. They represent 
their design as structured charts, use knowledge about 
structured development methodology, and heuristic guidelines.
Knowledge-based Design Assistant (KDA) incorporates a 
blackboard model of control for evaluating alternative 
designs and transforming specifications (in DFDs) into design 
(structured charts)[Shar88].

These knowledge based approaches, in general, use domain 
knowledge (knowledge about a particular subject or area) and 
knowledge about the methodologies to support the systems 
development activities. They do not include the 
organizational knowledge in their framework. They do not 
consider and reuse design-knowledge of other IS the system 
being designed will interact with. As a result, they are 
incapable of addressing some key issues organizations face 
today. They cannot ensure that specifications developed are 
appropriate for the environment in which the system will 
operate. They do not have knowledge of other systems and 
hence cannot ensure proper interface and integration. The 
problems of analysis and design redundancy and
inconsistencies become inevitable, which, in turn, obviously causes severe difficulty in maintenance.

5. Organizational Modeling and Organization Specific IS Development Support

Some approaches have attempted to model the information resources an organization possesses and to use the model during systems development. Prominent among them are three approaches which have some similarity with our approach of an organizational model. They are data dictionaries, information resources dictionary or repositories, and enterprise models. Underlying similarity among them is that they contain information specific to the organization.

Data dictionary contents generally include the schema of the databases it describes. A schema is the logical data-base description about the data items and their attributes [Mart77]. Data dictionaries contain constraints knowledge to a varying degree. Schema in a data dictionary represents an organization-wide data model of implemented systems if all IS of the organization are implemented on a single database, or if the organization has a centralized data dictionary. Data dictionaries enable reuse of data-model and data level integration if used during analysis and design stages (that is, if the development tools used by the organization are integrated with the data dictionary of the DBMS). It however
Information resource dictionary systems (IRDS) have an information resource dictionary (IRD) which is a central repository of data about information resources [Marc89]. IRD is a meta-database logically consisting of three types: entity, attribute, and relationships. Common entity types are data-entity types (data element, record, file, database, etc.), process-entity types (program/module, job, system, subsystem, etc.), and usage entity types (user, terminal, etc.). Enhancement of these systems to include knowledge models have been recently suggested [Hsu91]. One of the envisaged purposes include using them during applications development in a CASE environment.

As IRD contains information at the final level of implementation, substantial gap exists between the them and the software components at the analysis and design level. To that extent, they have similar limitation as data dictionaries. Also, they map more closely to the actual implementation of the systems than the real world, and do not encapsulate the real world object behavior. Moreover, IRD apparently plays a passive role during system development; it does not play an active role, like the proposed organizational model, and does not inherently enforces integration and reuse of organization specific software components.
Chen et. al. proposed an the use of integrated organization and information systems models (i.e., an enterprise model) in building and delivering business applications systems [Chen89a][Chen89b]. A model of extended critical success factors (CSF) method in [Chen89a] attempts to align organization's goals and strategies to its information systems by associating CSF measures with reports produced by information systems. An enterprise model presented in [Chen89b] consists of the organization structure, objective structure, information systems structure, and environmental structure, and interaction among four components. The issue of how to support, assist and manage systems development using them however has been left unaddressed. Also, issues of reusing past analysis and design and of enforcing integration are not addressed. Moreover, enterprise model is at a more abstract level than the organizational model we are proposing. Knowledge content of the enterprise model is captured in our approach in the organizational knowledge base.

6. Summary

To summarize, current knowledge-based approaches for supporting SD are not adequate for organizations of today. They do not (a) use knowledge of the organization to make the system suitable for the environment in which it will operate, (b) consider other IS the system being developed will
interact with, (c) reuse related past analysis and design information available in the organization to improve productivity and to avoid inconsistencies in designs, (d) provide a framework for integrated development of transaction processing and decision support systems, and (e) address the issue of maintenance when a single change affects multiple IS in the organization.

Current approaches that do use some organization specific information in systems development do not provide complete systems development environment and any support framework. Also, organizational information used is not sufficient for providing adequate support.

The OSD model is designed to eliminate these problems. Supporting in an organization environment, and providing unifying development environment for IS are among the key distinguishing features OSD model has compared to other knowledge-based environment.
CHAPTER III
THE ORGANIZATIONAL SYSTEMS DEVELOPMENT MODEL

1. Introduction:

Given the reliance of organizations on information technologies and problems associated due to lack of framework for systematic consideration and support of organizational perspective during information system (IS) development for an organization, we propose a model which alleviates these problems. We call this model organizational systems development model or OSD model. In this model, an organizational systems development environment (OSDE) is created using organizational knowledge and knowledge about other systems in the organization, and the system development process works on the organizational model. This approach allows us to address the reuse, integration and productivity issues in an integrated framework.

In this chapter, we propose the OSD model for systems development and maintenance with an organizational perspective. We discuss the processes and knowledge-bases and the resulting IS development environment. We also discuss theoretical foundations that justifies the proposed model and the development environment we call as organizational systems
development environment (OSDE), and how the proposed model address the reuse, integration and productivity issues.

2. The OSD model

As discussed before, IS for an organization needs to be developed with an organizational perspective to enable the system to be suitable to organizational goals, needs and practices. System development with an organizational perspective requires that:

a) all relevant organization specific characteristics are considered and incorporated in the functionality of the system being developed;

b) development process is constrained to enable the system under development to be consistent with the organizational goals, policies, procedures and practices; and

c) system being developed is also constrained to interface and interact smoothly with the other systems in place in the organization.

The primary objective of the OSD model is to enable IS development and maintenance with an organizational perspective. The OSD model supports the complete IS development life cycle, i.e., from accepting the user requirements to developing an implementable system and then maintaining it.
Figure 2: The OSD Model
Other objectives of the proposed OSD model include:

a) Provide explicit correspondence between the business processes of the organization, and the information systems that implement them.

b) Maintain explicit correspondence through various evolutions or changes that an implemented IS undergoes.

c) Assist an analyst in understanding the organization and obtaining a complete and correct set of requirements.

d) Support understanding of complex domains by providing knowledge of various domains and domain models along with reusable software components.

e) Provide an active meta information system, i.e., information system about IS in the organizations. Active in the sense that any change in an IS inherently gets reflected in the meta information system.

f) Support reuse of software components of all granularities during all stages of software development.

The OSD model has two main processes - specification development process and system-synthesizer process (figure 2). The specification development process supports developing system specifications from user requirements and propagating changes in the specifications when modifications are required. The system-synthesizer process generates an executable system from the specifications developed during the specification development process. These two processes
together constitute an organizational systems development (OSD) process.

We categorize the knowledge required to support the OSD process broadly as organizational knowledge, domain knowledge, system development knowledge, and organization model (which contains knowledge about other systems). These types of knowledge, together, provide the necessary environment for the OSD process to function effectively and meet the objective of systems development from an organizational perspective. We call the resulting environment as organizational systems development environment, or OSDE.

The organizational knowledge includes knowledge about the organization procedures, policies, guidelines, goals, planning and decision processes (and associated models), support systems in the organization, and the external environment. This knowledge becomes useful during the process of specification development from the requirements. The impact of using the organization knowledge is particularly high when developing cross-functional systems or reshaping business processes as the required knowledge is scattered in the organization.

The domain knowledge contains typical knowledge about various functional domains, like accounting, product planning, manufacturing, etc. This knowledge is generic in the sense that it is not tied to any particular organization. It includes knowledge of any domain of interest to the
organization. Thus, organization can make knowledge about decision domains like DSS and GDSS, or knowledge about models like statistical models and business models available during the IS development by including them in domain knowledge component. Domain knowledge assists in eliciting requirements from the users, validating the requirements, and reusing the specifications. Since domain knowledge is not specific to any organization, it can be exported and imported.

The **organization model** integrates the specifications knowledge of all the IS in place in the organization. Specifications of different IS are a subset (views) of this integrated specifications knowledge. We show later that use of organization model increases productivity and ensures proper integration of a new system with the systems in place. It exploits the underlying commonality between the IS to the maximum possible extent and enhances maintainability. If any modification affects multiple IS, the organization model enables identifying the systems affected, and making consistent changes in them.

The **systems development knowledge** contains methodology-related knowledge to support users during all the phases of system development process. Thus it has the knowledge required to validate specifications, organization model, views; knowledge to enforce analysis and design rules for the methodology being used; knowledge to transform specifications and generate an implementable system for a specified hardware
and software platform from the specification. This knowledge can also be exported and imported.

A key component within the OSD model is the organization model (OM). The organization model is composed of the integrated systems specifications of all information systems in the organization. Specific system specifications are subsets of the OM. We call the subset that defines the complete specification for a system as a view. This approach facilitates the integration and adaptability of systems within the organization.

The OSD process has traditional system development stages of requirement analysis and modeling, design and implementation. However, the underlying processes have some basic differences compared to other traditional and knowledge based approaches. These differences arise due to organization model component and organizational knowledge component. The OSD process focuses on utilizing existing specifications in the organization model and enhancing the organization model only if required. As a result, output of the OSD process is an enhanced, internally consistent organization model along with definition of the view of IS under consideration. In contrast, other approaches in general focus on developing the specification of the system under consideration in isolation and output is a standalone specification of that system.
3. The OSD process

3.1 Specification development process

The primary objective of the specification development process is to develop system specifications (what the system will do) from user requirements (what the user wants). Analysis and design of an IS is done using this process. We briefly describe the process here. As the IS development process in the OSD model depends on the OSDE, we discuss the process in detail after discussing OSDE.

During this process, user requirements are analyzed. The conceptual model of the system (very high level view of the system that corresponds to the 'real-world') is developed, and services the system should provide and constraints under which it must operate (like response time has to be less than 2 seconds) are established. Incompleteness and inconsistencies are identified and resolved. The process results in a complete, consistent specification of the IS. The resulting specification is stored in the organization model.

The process uses various types of knowledge to achieve its objective. It uses domain knowledge for understanding, exploring and evaluating options, and reusing analysis and design components. This high-level reuse of software components has similarities with the Draco approach [Neig84]. Our approach, though components-reuse based, accommodates refinement of high level specification using
knowledge from different domains (e.g. refinement of material issue specifications using accounting transaction concept from an accounting domain).

It uses organizational knowledge to fill the gaps in the requirements specified, and to ensure that specifications are consistent with the knowledge of the organization available in the organizational knowledge. For example, some exception conditions related to, say, loan approvals, if not specified by the users, get considered if available in the organizational knowledge. Process also uses organizational knowledge to resolve inconsistencies between user requirement statements.

IS knowledge in the organization model enables the process to access and reuse company specific software components in the system under development. The knowledge available is also used to ensure proper integration with the existing systems. The process acts as a knowledge acquisitioner as it updates the IS knowledge in the organizational model with the specifications of the system under development. IS knowledge is updated such that it does not have any internal redundancies and inconsistencies.

3.2 System Synthesizer Process

The objective of the system synthesizer process is to build an executable system package using the systems specifications developed during the specifications
development process. It transforms high-level specifications into working programs. It uses the system development knowledge base, and knowledge about the systems environment in the organization, which includes information about hardware, languages supported on the hardware, off-the-shelf packages available, database packages, and information about implementation of the other IS in place.

Some of the functions of system synthesizer process are:

1) Transform system specifications into working programs on the desired platform (hardware and software) by incrementally refining the specification until an implementation is derived.

2) Reuse, to the extent possible, program components for high-level software components that are reused. This avoids multiple implementation for identical specification, and enhance maintainability. Also, implementation decisions made before are reused. For example, try to reuse journal entry components from the accounting system when inventing an inventory system that makes journal entries.

3) Ensure technical compatibility with other IS in the organizations with which the system under implementation will interact and make implementation decision accordingly. The process uses IS knowledge and system development knowledge for this purpose. For example, a
journal transaction created by the inventory system should be 'readable' by the accounting system.

4) Ensure that the implementation decisions are compatible with the IS related policies and procedures organization has in-effect. These policies are available in the organizational knowledge.

5) Record or update implementation information in the IS knowledge.

This process is a key process in moving away from program-based maintenance to specification-based maintenance. Any changes required in the system is incorporated by changing the system specifications and then rederiving the implementation. Implementation decisions previously made are available in the IS knowledge and used during the re-implementation.

Three approaches commonly used for program synthesis are theorem proving, transformational implementation, and interactive programming environment [McC91]. In theorem proving, program is the side-effect when proving the statement

\[ \forall x \exists y [P(x) \rightarrow Q(x, y)]. \]

Transformational approach in general uses transformational rules to incrementally change specifications. Interactive

---

1Program is specified with input x, output y, P(x) is a set of conditions that hold on the input, and Q[x,y] is a set of conditions that must hold on the input and output after the program is executed [McC91].
programming environments address the computational explosion and search space explosion problems in the above two approaches.

The system synthesizer process in the OSDE model does not preclude any approach from being used. It, however, imposes additional conditions, derived from organizational knowledge and IS knowledge, that need to be satisfied to ensure that the resulting implementation is with the necessary organizational perspective. As the main focus in this dissertation is on the specification development process and the OSDE, system synthesizer process is not discussed in detail here.

4. Organizational Systems Development Environment

Researchers have widely recognized that software development is a knowledge intensive process, and have supported use of domain knowledge and software engineering knowledge for systems development [Aran85]. Various research projects demonstrate successful diverse use of these knowledge during systems development.

In addition to the above knowledge, we support the use of organizational knowledge and knowledge about other systems in the organization. These two type of knowledge address the organizational perspective problem. All four knowledge-bases together provide knowledge-based systems development
environment that we term as an Organizational Systems Development Environment (OSDE).

We model the OSDE as a four tuple,

$$\text{OSDE} = \langle K_0, K_d, K_{sd}, K_{IS} \rangle$$

where $K_0$ is the organizational knowledge, $K_d$ is the domain knowledge, $K_{sd}$ is the systems development knowledge and $K_{IS}$ is the knowledge about IS in the organization. In this dissertation, we concentrate on the organizational knowledge and on IS knowledge. Domain knowledge and systems development knowledge are addressed in various papers referred in chapter 2.

5. Organizational Knowledge

5.1 Requirements and the Role of Organizational Knowledge

Support for organizational knowledge (OK) is one of the key distinguishing features of the OSDE approach compared to other knowledge-based systems development environment. Supporting OK is particularly crucial for complex or interconnected systems as no single user may have the complete knowledge required, and knowledge required to be assimilated and incorporated in the specifications can be large. In this section, we discuss the role of OK, consider some constituent of OK, and discuss a model for knowledge about organization policies.

An analyst typically works on the requirements communicated by the user(s). Communicated requirements are
expected to be what the user really wants. These may be different than the actual requirements desired by the user. Moreover, these may be inconsistent with the organizational requirements, i.e., requirements that achieve the functionality and purpose the user envisages, but within the framework of organization policies, procedures, guidelines and objectives.

Requirements can be represented as a set of statements $s$ where $s_i = s_j$ if semantic content of $s_i$ is same as $s_j$. Let $AR$ be the actual requirements (set of statements) desired by the user; $SR$ be the requirements as stated by the user to the analyst and $OR$ be the organizational requirements. Ideally, $AR$, $SR$, and $OR$ should be identical. However, any of the following is possible:

(a) $SR$ may be incomplete and/or internally inconsistent.

Assuming $SR$ is internally consistent:

(b) $SR \supset AR$ (Requirements are overstated; may be crossing the boundaries of the system domain)

(c) $AR \supset SR$ (Requirements are understated)

(d) $SR - AR \not\subseteq Null$ and $AR - SR \not\subseteq Null$ (Some stated requirements are not in actual requirements, and vice-versa)

(e) $OR \supset AR$ (Actual requirements as perceived by the user are incomplete)

(f) $AR \supset OR$ (Functionality sought by the user is in excess of that required by the organization)
(g) $\text{AR} - \text{OR} \leftrightarrow \text{Null}$ and $\text{OR} - \text{AR} \leftrightarrow \text{Null}$. (Some actual requirements are not in organizational requirements, and vice-versa)

During the specification development process, analyst has the objective of making stated requirements identical with the organizational requirements. Using various types of knowledge, the analyst interacts with the users and evolves the requirements such that the inconsistencies between SR and OR keeps decreasing.

Let $\text{Limit}(x) \rightarrow y$ be the notation meaning that as the result of the $i^{\text{th}}$ iteration in the process, $x$ changes so that the gap decreases between $x$ and $y$, and as $i \rightarrow \infty$, $x \rightarrow y$. Then objective of the analyst will be to work on the stated requirements such that $\text{Limit}(\text{SR}) = \text{OR}$.

Modeling knowledge (MK), part of systems development knowledge, helps in overcoming representation related incompleteness and inconsistencies. Modeling knowledge and domain knowledge (DK) when used for requirements analysis during specifications development process achieve the objective of $\text{Limit}(\text{SR}) = \text{AR}$; that is, assist the user in stating the requirements so that the gap between stated and actual requirements keep decreasing.

SR can be both incomplete and inconsistent with respect to organization's policies, procedures, guidelines etc. Hence, to achieve the objective of $\text{Limit}(\text{SR}) = \text{OR}$, we require to use the organizational knowledge (OK). For example, for an
inventory system, OK can assist in enhancing the specifications to incorporate high-inventory warning signals based on company policies. In a loan appraisal system, it can assist in incorporating proper credit appraisal process.

We can thus specify the roles of the three types of knowledge as follows:

(a) Apply (MK;SR) to achieve complete, consistent SR
(b) Apply (MK,DK;SR) to achieve limit(SR)=AR
(c) Apply (MK,DK,OK;SR) to achieve limit(SR)=OR

where (c) can be read as: apply MK, DK, OK on stated requirements to achieve the objective of Limit(SR)=OR.

Although (a) and (b) can be achieved by MK and DK, we do require OK for achieving (c). For example, when developing an accounting system, DK can ensure that only the allowed depreciation methods are used. OK is required to ensure that, of the allowed methods, one used by the company is selected.

5.2 Organization knowledge as a collection of knowledge

OK is further organized as a set of different types of organization specific knowledge. These together act as the repository of the current status of the knowledge accumulated over a period. Knowledge about models, policies and procedures, and external environment, are all part of OK.

We view organizational knowledge as

\[ K_0 = \{K_0.p, K_0.m, K_0.se, K_0.ISp, K_0.ext, \ldots \} \]
where $K_{o.p}$ is the knowledge about policies of the organization, $K_{o.m}$ is the knowledge about various models used in the organization, $K_{o.se}$ is knowledge about the systems environment that exists in the organization, $K_{o.ISp}$ is knowledge about the systems development policies the organization has, and $K_{o.ext}$ is knowledge about the external environment of the organization.

Each of these knowledge may have an underlying model according to their individual characteristics. We present here a possible model for the organizational policies knowledge.

Organizations have policies as a means of management and to meet its goals. Policies are organized in a hierarchical fashion. Lower level policies refine and implement policies at a higher level. Lowest level policies are completely in terms of actions/procedures; i.e., goals expressed purely in terms of operations. Policies may conflict, and may apply only under some conditions. We model the policies (adapted and enhanced from the modeling of policies as done by Moffett & Sloman [Moff91]) as a collection of tuples:

$$<P,T,S,O,G,C,Pp,Pc,Oc,Cp,Cpb>$$

where

- $P$: policy name
- $T$: policy type
requiring/permitting/forbidding/deterring²

S: policy subjects (to whom it applies)
O: policy objects (at which it is directed)
G: goal (operation/action on target object)
C: constraints (predicates to be satisfied before policy acts)

Pp: parent policy (policy that is being refined/implemented)
Pc: child policies (policies enacted to implement this policy)

Oc: organization of child policies:
    collection (order is unimportant, all child policies need to be satisfied)
    order sequence (child policies need to be satisfied in the specified order)
    or (at least one of the child policies need to be satisfied)
    precedence-or (specifies preferred sequence for satisfying 'or' child policies)

Cp: conflicting policies that precede this policy
Cpb: conflicting policies that do not precede this policy

The hierarchical organization of the policies knowledge-base enables the analyst to explore the relevant branches.

²Refer [Moff91].
This organization also benefits in propagating modifications in the policies KB and in identifying systems that require changes when a policy changes as a change in a policy node affects all policies below that. Some examples of the policies that illustrate the suitability of a hierarchical organization are:

- **Maintain adequate inventory levels**
  - Reorder whenever inventory is < 2 months consumption
  - Reorder only if item is required in the next month
  - Release purchase order at the beginning of the month

- **Reward customers that are loyal to the company**
  - Consider customers loyal if they purchased regularly for last 2 years
  - Consider customers loyal if they purchased goods worth $6000 in last 6 months
  - Send special coupons
  - Send New-Year cards

### 6. IS knowledge and Organization Model

In an organization a system rarely operates in isolation. Developers need to identify all systems that will interact with or relate to the system under development. They need to develop clear and correct understanding about these interactions and interrelations; and evaluate implications of their decisions on other systems. Errors of omission and inadequate understanding may result in costly delays and inconsistent systems, especially for large and complex systems. Moreover, some components of the system under development may have already been analyzed as a part of some other system in the organization. Such components are
specific to the organization and should be reused. For example, analysis of 'pricing components' can be effectively reused for a company when developing a product planning system.

During systems development process, a system model that closely approximates the real world is built "to understand the behavior of the system under study; to replicate the system functionality in a computational environment; .." [Sara92]. This model in general will have entities, with its structural and behavioral characteristics, along with relations and interaction among entities to achieve certain goals.

Suppose a system model \( M \) is represented as a set of various components in the model. If \( M_{new} \) is a model of a system being analyzed and \( M_{old} \) is a model of a related system developed before, then \( M_{new} \cap M_{old} = \emptyset \). Reusing \( M_{new} \cap M_{old} \) from \( M_{old} \) should decrease effort in analysis, and ensure consistency of analysis between two systems.

We propose an organizational model (OM) as an integrated model of all systems in the organization. It forms the basis of recording and representing the integrated results of all analysis (and design) done for various systems in the organization. If \( S = \) set of all systems implemented and used in the organization, and \( M_i \) is a model of the \( i^{th} \) system and \( M_i \in S \), then we consider the organization model to be:

\[
OM = M_1 \cup M_2 \cup M_3 \cup \ldots \cup M_n \quad \text{where} \quad n = |S|, \quad M_i \in S, \quad i=1,..n.
\]
We model the organizational model, underlying model for KIS, as

$$OM = <E, A, M, C, F, O_p, O_c, R, D, S, I>$$

where

- **E**: entities
- **A**: attributes
- **M**: behavioral characteristics or methods
- **C**: constraints
- **F**: facts
- **O_p**: parents
- **O_c**: entities which are specialization of this entity
- **R**: relations with other entities
- **D**: design objects associated
- **S**: IS using this entity
- **I**: implementation details for this entity

The organizational model thus subsumes all system models in the organization; and any system model in the organization is a subset of the organizational model. For any particular system of interest, presenting only the components that are part of the system, and external linkages of these components, effectively presents the system model from the organizational model. We consider system models for various systems as **views** from the organizational model. This approach has similarities to the views in relational databases [Maie83].

With an organization model of the organization in place, analysis of a new application will entail a) identifying and using $$M_{\text{new}} \cap OM$$; and b) extending $$OM$$ by $$M_{\text{new}} - OM$$, so that $$OM \supseteq M_{\text{new}}$$ (see figure 3). We show in the next chapter that, an organization model, if implemented correctly, can ensure proper integration of a new system with the systems in place.
in the organization, enable high level specification reuse, increase productivity, and exploit the underlying commonality between the IS to a maximum possible extent. Also, incorporating a change in any element of OM can provide consistent change to all tasks (possibly in multiple systems) using that element; and can make the change available at the same point of time.

Organization model offers some additional advantages. It is an active information base about various systems in the organization. It is active because any change in a system simultaneously gets reflected in the OM. OM can enhance the understanding of the analyst and assist the communication between the analyst and users.

A distinguishing feature of the organization model compared to other approaches regarding reusability is that it supports direct reuse of organization specific specifications. This compares favorably with reusing software components from domain knowledge where the components may have to be modified to incorporate any organization specific characteristics.

7. Systems development in the OSD model – an overview

In this section, we overview the process of information systems development with an organizational perspective using the OSD model. We discuss the role of various OSDE model components and their effect on the system being developed.
Broadly, IS development in the OSD consists of three stages: requirements analysis and design (done during the specification development process), and implementation, which is done during the system synthesizer process.

The development process in the OSD model is highly interactive. On the basis of the user requirements, a view is extracted from and validated against the organization model. If the required components are not present in the OM then the extracted view is enhanced and ensured to be complete and consistent with the OM.

Figure 3: OM-based application development model

The specification development process focuses primarily on managing the view and enhancements in OM. As the existing components are reused, this approach enables direct reuse of organization-adapted software components. As the new components or enhancements to the existing components are
ensured to be compatible and consistent with the OM, this addresses issues related to knowledge of other IS in the organization, and integration issues. As no software component is duplicated in the OM, any modification in a software component will result in consistent modifications in all systems using that component.

Along with managing the view, during the specification development process, a view is checked against the organizational knowledge for consistencies and completeness. Organizational knowledge is accessed for knowledge related to entities being worked on, and the view is enhanced to take care of organizational requirements not specified by the user. Arbitration is required if organizational knowledge-base conflicts with the user requirements.

During the specifications development process, domain knowledge is actively used to prompt the user for possible enhancements, to reuse any possible software components either directly or with modifications, and to provide knowledge about the domain to the user. For example, previously developed design objects from the domain knowledge-base can be reused for enhancing the view.

After the complete view of the system is developed, system synthesizer process builds an executable system package. It transforms high-level specifications into working programs. It uses the system development knowledge base, and knowledge about the systems environment in the organization,
which includes information about hardware, languages supported on the hardware, off-the-shelf packages available, database packages, and information about implementation of the other Is in place.

8. Summary

Today's organizations typically require their changing information system needs to be addressed quickly, and reliably. They also require information systems that are integrated horizontally (e.g., integration among transaction processing systems) and vertically (e.g., integration among transaction processing and support systems). In this chapter, we proposed an Organization Systems Development Environment model that addresses the organizational perspective problem in systems development and maintenance, and enables information systems development for an organization from an organizational perspective.

The approach we adopt is innovative in many ways. We address the information system development and maintenance from an organization perspective to enhance the suitability and reliability of an information system. Besides the knowledge about organization, knowledge about specifications of all IS in place in the organization is used through the organization model to provide significant integration, adaptability and reuse capabilities.
Developing the OSD processes, and implementing OSDE in an organization are complex tasks. For a successful OSDE, organizations require organization KB and organization model. Building them requires substantial investment of time and money. However, as it is crucial that organizations adopt OSDE for competitive edge, organizations can adopt the strategy of implementing OSDE and start incrementally building the required KBs.
CHAPTER IV
ORGANIZATION MODEL and IS DEVELOPMENT

1. Introduction

In chapter III we defined organizational model (OM) as an integrated specification model consisting of specifications of all information systems (IS) in the organization. We also briefly discussed its impact on IS development. Critical feature of the OSD model is the OM and OM-based IS development. In this chapter we discuss in detail the IS development process using OM and application views, and representation issues associated with the OM to maximize reuse.

2. Representation of OM

Requirements for an information system problem can be specified in a large number of ways [Wing88]. Different specifications for the same problem, in general, may have different suitability with respect to integration, reuse, maintenance, and productivity. Considering the demands placed on IS today, it is crucial that specifications are represented in the most beneficial way. We discuss below some desired characteristics a specification representation should
have to enable an organization to respond quickly, and to maximize the reuse of specifications.

2.1 Evaluation criteria

Specifications broadly consists of two parts - model of the information system (IS) that approximates the 'real-world', and specifications of the tasks performed by the system. Let IS specification universe (A) for a given problem be a set of all possible specifications for the IS. Each of these specifications are combination of a model and task specifications that specify the system completely. Let IS model space (M) be the set of all possible models that approximate the system; and task specification space (T) be the set of possible task specifications sets for the system. Then A, M, and T can be formally defined as:

\[ M = \{ m | m \text{ is a possible model of the system} \} \]
\[ T = \{ t | t \text{ is a set of task specifications for the system} \} \]
\[ A = \{ (m, t) | (m, t) \text{ specify the system completely} \} \]

where \( m \in M \) and \( t \in T \).

Elements of M and of T will have varying amounts of IS knowledge. But the specification of the system, \((m, t)\) assuming it is complete, will have the relevant IS real-world knowledge required to develop the IS. If \( K(x) \) denotes the knowledge content of \( x \), then \((m, t) \in A \) if \( K(m) \cup K(t) = K(\text{relevant knowledge of the system real world}) \).
Definitions:
1. \((m, t)\) is a minimal specification if \(K(m) \cap K(t) = \emptyset\) and \((m, t) \in A\).
2. Model approximation factor (MAF) is a fraction of relevant knowledge of system real-world captured in the model of the information system.
3. Task knowledge factor (TKF) is a fraction of relevant knowledge of the system real-world captured in the task specifications for the information system.

Claim 1: For a given minimal specs \((m, t)\), increase in MAF \(\iff\) decrease in TKF.

Proof: \(K(m) \cup K(t) = K\text{(relevant knowledge of system real world)}\) and \(K(m) \cap K(t) = \emptyset\) (minimal). If \((m_1, t_1)\) is another minimal specs for the same system, then \(K(m_1) \cup K(t_1) = K(m) \cup K(t)\).

If \(K(m_1) - K(m) = \Delta K(m)\) then \(K(t) - K(t_1) = -\Delta K(m)\)

\(\implies\) if \(K(m)\) increases by \(\Delta K(m)\), then \(K(t)\) has to decrease by \(-\Delta K(t)\) for \((m, t)\) to be minimal.

Claim 2: High MAF and low TKF will result in a system that has high maintainability.

Proof: Let \((m, t)\) be the minimal specs.

Let \(t\) be a set of tasks \(\{t_1, t_2, \ldots t_n\}\).

Let \(k = K(t_1) \cap K(t_2) \cap \ldots \cap K(t_i)\) be the common system real-world knowledge of the \(i\) tasks (first \(i\) tasks without any loss of generality), \(i < n\). (For low TKF, \(k\) will be small; empty if
Claim 3: High MAF and low TKF will result in high reusability, and low analysis effort.

Proof: Let (m,t), k, and (m1,t1) be as in the previous proof. Suppose the relevant knowledge of the system real-world does not change but the user requirements increases (for example, an additional report is required).

Suppose the additional task uses k.

Then incorporating this new task in t will require reanalyzing the system domain for k, and encoding it in the new task. Hence, more effort is required if (m,t) is used. Also incorporating k in new task exposes the system to the risk of incorrect
incorporation of $k$, and lower maintainability (claim 2).

New task in $t_1$ will reuse $k$ already available in $m_1$ => lower risk of errors, less effort, higher maintainability, higher reuse.

Thus specifications should be represented such that MAF is very high, and TKF is low. An ideal representation will have MAF=1, and TKF=0.

2.2 Comparison of some common representations

Various methods have been used for representing system domain models. We examine four popular representation techniques: data flow diagrams, entity-relationship diagrams, semantic data modeling, and object-oriented modeling.

Data flow diagrams (DFD) are based on data flow approach. The model is represented as data-flows and bubbles (representing processes) to approximate the real world. DFD representation method is popular in structured analysis. The data flow approach has strong functional emphasis and weak data structure emphasis [Coad91].

Entity-relationship (ER) diagrams, and semantic data modeling (SDM) are the information modeling approaches. ER diagrams give relationships between the entities of the real world, and attributes of the entities and relationships. Semantic data modeling also gives attributes and relationships of the entities or objects in the real world;
it additionally gives structural relationships between the objects.

Object oriented model (OOM) represents objects of the real world directly in the model. Along with the attributes of the object, it also represents behavior of the objects and supports inheritance of the behaviors and communication with messages.

We compare four representation techniques: DFDs, ER models, SDM, and OOM. Let

- $O_A$ be the set of objects where each object is a set of attributes;
- $O_{AS}$ be the set of objects where each object is a set of attributes and methods/services;
- $M$ be the set of messages between objects required to provide services;
- $S$ be the set of methods/services (both procedural and non-procedural);
- $A$ be the set of attributes;
- $ST$ be the set of structural relationships (generalization-specializations; whole-part) between objects;
- $AS$ be the set of associations or relations (non structural) between objects.

Then $OOM = \{O_{AS}, M, ST, AS\}$; ER model = $\{O_A, AS\}$; SDM = $\{O_A, ST, AS\}$. DFDs can be approximated as $\{O_A, S\}$. Thus $OOM \supseteq SDM \supseteq ER$ model, and $OOM \supseteq DFD$. Hence, $K(OOM) > K(SDM) > K(ER$ model), and $K(OOM) > K(DFD)$; where $K(OOM)$ means knowledge about system.
domain captured in the OO model. It follows from the discussion before that $\text{MAF(OOM)} > \text{MAF(SDM)} > \text{MAF(ER model)}$ and $\text{MAF(OOM)} > \text{MAF(DFD)}$. We can conclude that among the four representation techniques, OOM is the preferred representation on the basis of the criteria developed.

3. Object-Oriented Representation for OM

The above section compared four representations for modeling the system domain, and showed that the object-oriented model (OOM) was superior compared to others. Compared to other representations, OOM maps the real world more directly [Coad91]. Along with the structure, OOM also captures the behavior of the real world objects. Hence, to capture the semantics, functionality, constraints, behavior, and associations of the organization objects, OOM is the logical choice. We give a brief overview of the OOM and then give an object-oriented representation scheme for the OM.

An object is a concept, abstraction or an entity of something in a system domain. All objects have identity, state and a defined set of methods (also referred to as services or operations) that modify that state. An object class describes a group of objects with common attributes, methods and semantics. An attribute is a property of the objects in a class; a method is an action that may be applied to objects in a class. The state of the object may not be accessed except via these methods. Consequently, objects
communicate by passing messages to each other and these messages initiate object operations. Message consists of object identifier, name of desired service, and parameters.

Objects can have aggregation relationship (whole-part, or is-part-of) with other objects. For example, assembly is aggregation of components; set is aggregation of elements. Classes can have hierarchical structures through generalization (is-a) and specialization. A generalized class is an abstraction of similarities among similar classes. Vehicles is a generalization, hence superclass, of cars and trucks. Refinements of a generalized class are the specializations. Specialized classes inherit the features of its superclasses. A subclass, however, may override a superclass feature.

Objects can have associations with other objects. An association describes a link and common semantics between two objects. For example, is-author-of associates author with book. Associations are inherently bi-directional. An object model describes objects, classes, generalization - specialization and whole-part hierarchies, and associations for the system domain of interest. Object-oriented methodologies are discussed in detail in [Coad91, Rumb91, Booc91].

Figure 4 presents a possible object representation scheme for the OM. Encapsulated within the object are non-procedural services, assertions and rules for enhancing the knowledge content. Non-procedural services can include
Figure 4: Object representation scheme for the OM
services based on production rules, inference mechanisms or some other AI technique. For example, it can have rules to classify a loan application. Assertions and rules capture some knowledge about the object that can be used for, say, decision processes or requirements elicitation.

Triggers indicate the actions to be undertaken when some conditions become true. They are like: if <condition> then <action>. Triggers can be associated with creation, modification or deletion of the object, or other attribute of an object.

Object-oriented model using the above representation enables substantial knowledge of the object to be encapsulated within the object. MAF for this representation is high, and using this representation is a step toward meeting the needs of a dynamic organization.

3.1 Object-oriented meta-model for OM

Figure 5 shows the underlying object model to capture the object-oriented representation for the organization model. It is a meta-model as it is a model of a model. It considers the organization-model as top-most class. It consists of a collection of Objects. Each object is considered as a collection of Attribute objects, Services objects, and Relation objects. The relation class has three specializations, namely, Parent-child class, Part-of class, and Instance class. The relation object requires at least two Object objects.
Application-view objects are part of Organization-model objects. Application-view objects relate to Object, Attribute, Services, and Relation objects, as view object refer to these objects for defining the view. This relationship is not shown in the diagram for clarity.

Each of these classes contain provision for capturing detailed information about the object they represent. For example, Attribute has information about type (numeric, string, etc.), whether calculated, valid ranges, conditions it should satisfy, etc.

![Organization Model Meta-model](image)

**Figure 5:** Organization Model Meta-model
An inventory object in this meta-model will be stored as Object Inventory; attributes name, number, quantity available as Attribute objects Name, Number, Quantity_available. If special-inventory class is specialization of (child of) Inventory, it will be stored as Object Special-inventory, and Parent-child relation between Inventory and Special-inventory to inherit Inventory characteristics.

3.2 Object categories in OM

The organization model has complete details about analysis, design and implementation. Objects in OM represents results obtained at various stages of IS development. We structure these objects as follows.

1. **Analysis objects**: Objects that result from requirements analysis of applications. They in general correspond to objects or entities in real-world, and relations among them. These objects define classes in the applications. For example, an employee object corresponds to the employee class in the organization.

2. **Task objects**: Objects that specify sequence of operations on objects using their methods (or services) to achieve the objective of the task. A set of tasks achieves application objective.

3. **Generic design objects**: Underlying design objects that organization uses for representing design of analysis objects. Examples include user-interface objects like report objects, graphic objects, window objects; data
manipulation objects like database objects; external package objects like spreadsheet objects; communication objects. These objects together constitute collection of design knowledge in the organization over a period of time.

4. **Application design objects**: These objects describe the design decisions made for analysis objects. These are instantiations of generic design objects that are chosen for designing analyzed objects.

![Diagram of Sales report and generic objects](image)

**Figure 6: Sales report and generic objects.**
5. **Implementation objects**: These objects relate task objects and application design objects to actual implemented objects (code).

6. **Application-view objects**: Objects that describe views of various applications in the organization. For example, application-view object order-confirmation-system describes analysis objects, application design objects, task objects, and implementation objects that constitutes the application order confirmation system. Figure 6 shows an example of an analysis-object, sales-report, and related other types of objects.

4. **View and OM Related Algorithms**

As discussed before and in chapter 3, one of the key activities during IS development and maintenance in OM-based approach is in managing the view and the OM. User or analyst interacting with the system is working on the view; operations on OM are done in the background and not under direct control of the user. In this section, we discuss various operators required to manage views and the OM.

**Create View**: This operator creates a new application-view object for the new IS. A newly created view will not have any object from the OM in its view. It may have information about application objective, coordinator, etc..
Generate view: This operator generates the view from the OM, and makes it available in the workspace (or working memory) for working. A pseudo-code algorithm for generating view is:

\[
\text{Generate-view(view):}
\]
\[
\text{Begin}
\]
\[
\text{Select Object.object}^1 \text{ if it is in the view}
\]
\[
\text{For every object selected,}
\]
\[
\text{select Attribute.objects if in view,}
\]
\[
\text{select Services.objects if in view,}
\]
\[
\text{select Relation.objects if in view}
\]
\[
\text{end}
\]

At the end of view-generation process, a subset of OM, with underlying structure same as OM, is generated and made available in the workspace.

Enhance view: Enhancing a view includes adding a new object, or adding an attribute or service to an existing object in a view, or adding relationship between two objects in a view, or adding a constraint additional characteristic to any these. View enhancement is a critical operation as it initiates OM enhancement if required; and it

\[^1\text{X.y signifies object y of class x.}\]
directly affects reuse and integration. A pseudo-code algorithm for enhancing view is:

Enhance-view (view, enhancement-needed)

Begin

Search for enhancement-needed in OM

If search successful, present search results for selection

If any selection made,

  include the selection in view,
  present related objects for consideration and selection
  Initiate Enhance-view (view, additional-selected object)

If search unsuccessful, or no selection made,

  initiate Enhance-OM(view, enhancement needed)

End

Enhance-view operator calls enhance-OM operator if required enhancement is not available in the OM. A pseudo-code algorithm for enhancing OM is:

Enhance-OM(view, enhancement-needed)

Begin

Search for needed enhancement in the OM

Continue further if search unsuccessful or unsatisfactory
Search for needed enhancement in domain knowledge

If search successful,

validate candidate objects against org. knowledge

present search results for selection in decreasing order of fit; along with misfit details

If any selection made,

include the selection in view,

present related objects for consideration and selection

Initiate Enhance-OM (view, additional-selected object) (optional; can be under user control)

If search unsuccessful, or no selection made,

accept new object specification from the user.

For any new object selected or entered:

validate the new object definition against org. knowledge.

link the object with org. policies where applicable

validate enhanced OM

End

Modify view: This operator is used to change any information in the view of the application. Modify view operator is needed if an object, an attribute, a service, or a relation needs to be modified or removed. A pseudo-code algorithm for modifying view is:
Modify-view (view, modification)

Begin
validate modification in view
validate modification with organizational knowledge
identify all affected components in the OM
validate modifications in all affected components
if modification validated and accepted,
mark all affected objects for modification
end

Validate view and validate-OM operators validate the view
and OM using systems development knowledge and
organizational knowledge. These operators record
validation failures as pending issues for user to
address them.

5. OM-based IS development process

The underlying model of OM-based IS development was
presented in chapter III (refer figure 3). The IS developer
is expected to proceed along the basic steps listed below for
developing a specific application using an OM.

1. Create or generate view.

OM-based IS development process starts with
creating a view for a new application. This view will
have basic information about the application like
objective, coordinator, location, etc. View for the
application will already exist if the user has been working on developing the application or if an implemented application is being worked upon for maintenance. In such a case, developer generates the view for the application.

2. Select (from OM) objects required and available in the OM. Objects selected from OM provide reuse of objects that are adapted to the organization needs during previous development efforts. These objects will also be linked to organizational polices if polices knowledge is in the OSDE of the organization.

3. Enhance objects in the view.

Objects in the view are incrementally enhanced to meet the needs of IS under development. If enhancement is available in OM, it is selected. Otherwise, enhancements are first looked for in the domain knowledge. If not available, they are created as new objects. In either case, it results in enhancement of OM, and these objects are available for future development tasks.

4. Modify objects in view if necessary.

Modifies objects in view will modify OM also. If an existing system is undergoing modification, some objects will definitely need modification. Modification impact on other existing systems and validity with organizational policies is evaluated before making modification permanent. For a new application,
modification of existing objects is not expected.
(Enhancement of existing objects, in term of attributes,
additional methods, etc. can be expected.)

5. Create/modify task objects.

Manipulation of objects using their services is
specified for each task in the application.

Figure 7: Objects creation in OM-based IS development

6. Create/modify application design objects.

Application design objects corresponding to
analysis objects in the application are created by
instantiating appropriate generic design objects. In
this step, design knowledge and expertise developed and
accumulated over a period of time is used. During this
step, design decisions made for application under
development get recorded. Also, previous design
Figure 8: System development processes in OM-based IS development
decisions made for objects used in other applications are available for reuse.

7. Transform design and task objects into programs.

As all objects in the view have object-oriented representation, disadvantages associated with changing of representation from analysis to design are absent; and developer can move back and forth between analysis and design.

Figure 7 gives the order in which various objects are created during IS development using OM. Figure 8 gives an overview of the interaction of various processes involved in the OM-based systems development.

6. Advantages of OM-based IS development

OM-based IS development process suggested here has several advantages over traditional methods and other knowledge-based IS development processes. Our approach, if implemented properly, enhances proper integration, maximizes reuse, enhances productivity, and decreases development time. Moreover, our approach address the problem of maintenance of existing application.

Integration: OM consists of all analysis, design, and implementation objects in the organization. OM has no redundancy. Hence, an object used in multiple applications is identical in all such application as views of these application refer to the same object. Hence, our approach inherently supports integration, and
Integration is supported during all three stages: analysis, design, and implementation.

**Reuse:** An object if available in OM is reused and is not created. This maximizes the reuse of available components in OM. If a component is not available in OM, but available in the domain knowledge, it is reused from there. Hence reuse of both organization specific and generic components are maximized.

From the reuse perspective, OM-based IS development model is given in figure 9. This model is a significant improvement over the reuse model in [Isod92] as our model allows richer set of reusable and directly usable components.

![Figure 9: Reuse-oriented OM-based IS development model](image)

**Figure 9:** Reuse-oriented OM-based IS development model
OM-based development maximizes reuse during different stages of development. Two applications may share common analysis objects, but may have different application design objects corresponding to these analysis objects. Hence common analysis objects get reused, while application design objects are different.

**Productivity:** All objects reused from OM are already adapted and validated. Effort, time and errors for analyzing and developing these products is saved. Also, OM enables better understanding of the requirements and interaction with other IS in the organization. All these enhance productivity.

**Capture of design knowledge:** OM-based development captures design knowledge in application design objects, and knowledge about related analysis and implementation objects.

**Flexibility in making design changes:** OM based approach enables flexibility in design changes and both micro (design change for an object) and macro (design change across applications) levels.

At micro level, design changes are easily incorporated by instantiating a different design object. For example, if an x-y graph is to be changed to a bar chart, generic bar-chart design object is instantiated for the analysis object. At a macro level, generic
design object is changed. This changes the design of all its instantiations.

**Specifications based maintenance:** All modifications are performed by identifying changes required in the OM. All implementation objects associated with any object that has changed are recreated by transformation. Need for maintenance on the code is avoided.

**Consistent implementation of changes across IS:** As changes required are localized and changed in the OM, followed by redriving of implementation, all applications affected by the change are identified and changed simultaneously and consistently.

**OM as software information system:** As OM has complete information about all applications, and this information is always current (given the active role of OM in ID development), it becomes an excellent source for information about IS in the organization.

### 7. Summary

In this chapter we developed two criteria, model approximation factor and task approximation factor, for evaluating a representation scheme for the organization model. The criteria is based on maximizing the knowledge content of the system model that maps real world as closely as possible. Based on our developed criteria, we compared various representation scheme like data flow diagrams, entity relationship diagrams, object-oriented representation etc.,
and showed that object-oriented representation is the most suitable representation for OM.

We next proposed a meta-model for the OM that enables object-oriented representation of the specifications in the OM. We categorized various objects in the OM as analysis objects, generic design objects, application design objects, task objects, implementation objects, and view objects. We presented algorithms to generate views, and to perform various operations on views and on OM.

We next discussed the OM-based information systems development processes and stages, interaction among the processes and order in which various objects are developed. We presented steps that a developer is expected to follow in the OM-based IS development. We concluded the chapter with the discussion on benefits, showing that our OM-based IS development approach inherently supports integration, reuse, enhances productivity, and addresses various issues of direct concern to organizations.
CHAPTER V
ARCHITECTURE OF THE OSD MODEL

1. Introduction

Given the OSD model as discussed in chapter III, the issue arises as to what should be the architecture that implements the model and achieves its objectives. An architecture for the OSD model can be expected to be complex as it has to support retrieval, updation, exploration and management of organizational, domain and systems development knowledge; create OSD environment and ensure that specifications are consistent with the organizational knowledge; and manage the organization model (and views), and ensure that it is internally consistent with no redundancies, and that its integrity is maintained.

It should also support the underlying processes that support reuse, link specifications to organization knowledge components like business policies, relieve the IS developer from mundane activities and enable him to devote his efforts to complex and intellectually challenging aspects of system development, provide IS knowledge base, and enable fast and consistent maintenance of systems.

In this chapter we present and discuss a knowledge-based architecture of the OSD model. Knowledge-base is database
augmented with reasoning mechanisms. Knowledge-based architecture is a logical choice as significant amount of diverse knowledge need to be managed and used. As our concentration is on specification development process, we also present and discuss its possible architecture.

2. Architecture for the OSD model

The architecture we propose for implementing the OSDE model consists of six basic building blocks - specifier, system synthesizer, domain knowledge bases (DKB), organization specific knowledge bases (OKB), IS knowledge base (ISK) and systems development knowledge base (SKB) (figure 10).

Specifier in the architecture provides functionality of the specification development process of the OSD model. It consists of collection of tools for a system developer to use. Synthesizer generates the information system package that is specific to the systems environment (hardware and software environment) in which the package will operate. Development or maintenance of a system is initiated by interacting with specifier.

DKB, OKB and SKB are collection of knowledge-bases of their respective categories. As these knowledge-bases are used by the processes but not updated by them (thus one way relationship), their maintenance is considered beyond the scope of the proposed architecture. They are maintained by knowledge engineers. This one way relationship has advantages
as this enables portability of DKB and SKB among OSD environments and high cost of developing DKB and SKB can be distributed [Neig84]. Knowledge engineers maintaining and enhancing DKB and SKB need not be internal to the organization.

OKB however is not portable to another OSD environment as it is organization specific, applicable only to the organization in which it exists, is built over a period of time, and is maintained by the knowledge engineers. These knowledge engineers, in general, are from within the organization.

Responsibilities of knowledge engineers working on the three knowledge-bases differ. Knowledge engineer for DKB has the responsibility of encoding knowledge or expertise, and developing library of reusable components of varying granularity for the domain under consideration. His role as envisioned here encompasses the role of application engineers in ITHACA [Fugi92] and domain experts - domain analyst and domain designer in Draco [Neig84].

Knowledge engineer for SKB focuses on encoding knowledge about system development methodologies, heuristics, guidelines; program generation and transformations, capabilities and characteristics of various hardware and software platforms, etc. Knowledge engineer for OKB continuously updates and enhances the OKB and ensures that it is up-to-date. This is critical as, as discussed before, OKB
directly affects the specification of system under consideration.

Since DKB, SKB, and OKB are created and maintained outside the scope of proposed architecture, it becomes imperative that they are based on a representation scheme that OSD process can use; and they provide access mechanism that the OSD process understands. As the OSD process uses object model, and organization model is based on an object model (see chapter IV), it logically follows that representation for DKB, SKB, and OKB should be object oriented. Object oriented representation also facilitates reuse and managing of complexity.

The proposed architecture provides knowledge-based intelligent support through complete system development life cycle, including maintenance. It is unusual to find an architecture that support these three knowledge-bases, IS knowledge-base and processes that support IS development at all stages. Programmer's Apprentice [Rich88] support all stages, but it lacks organizational perspective, and is based on waterfall model. Architecture of IDeA supports design stage only and does not support organization specific knowledge [Luba89]. ITHACA architecture is designed for high reusability of software components and uses object-oriented techniques and software information base; but it also lacks organizational knowledge support [Fugi92].
Figure 10: Architecture for the OSD model
3. Specifier

Specifier is a key component in the OSD architecture. It performs the functions of and achieves objectives of the specification development process. Most of the activities related to systems development and maintenance are done by this process.

Specifier provides inherent capability to support integration, and high level reuse of analysis and design information. It manages internal representation of specifications and the organization model, and provides quick response capability to handle changes. For modifications, it identifies and propagates changes to all IS affected. It controls and provides support for the system development activities at all stages, and enables users to use the knowledge from KBs.

Specifier is designed to interact with analyst and act as an assistant to the analyst. Other architectures have taken similar approach in the sense that the interaction is with the analyst, and not directly with the users; e.g. Requirement Apprentice [Reub91]. As the environment in which the specifier operates is rich in knowledge and compensate for lack of expertise or knowledge of the user, we envisage that specifier can effectively interact with users with a front-end natural language interface.

Most of the activity with Specifier relate to managing the application view and IS knowledge base (which is based on the organizational model) from which it gets created.
Specifier maintains the application view of current application user is working on along with extensions that need to be made to the organization model. It matches user requirements with the organization model, and includes available components in the view. If any components is not available, it matches the requirements with components in domain knowledge base. If it is available in the DKB, it evaluates the component for inclusion in the view and for extension of the organization model. Specifier thus acts as a knowledge acquisitioner for IS knowledge base and ensures that views and the organization model are internally consistent and without any redundancy.

Besides managing the application view and organizational model, specifier interfaces with DKB, SKB, and OKB; has in-built reasoning mechanisms to navigate through large amount of knowledge, validate views and organizational model, and assist user in the development process. As such, it is expected to be very complex. To manage the complexity and to have flexibility of incorporating functionality in a phased manner, we propose a modular component based architecture for specifier. A possible architecture envisioned at this point is shown in figure 11. We discuss below the function of each component.

**User interface:** The user interface provides flexible interface for the users to interact with the system. The interface is capable of adjusting to the interaction and skills levels of the user. A skilled user or an analyst can
initiate the activities by directly choosing or issuing a command. A novice may seek advice for understanding the options available and in such case, advisor component may be very active.

User interface provides users the ability to switch between different operating modes. For example, the system may function in an advisory mode, validation mode, refinement mode, browser mode, options explorer mode, elicitation mode, evaluation mode and view mode.

**Controller:** Controller, as the name suggests, controls all the activities of the specifier. It coordinates the activities of different components, manages the context switching by the user among different modes, and actively participates during the specification development process by initiating evaluation and validation modes in parallel to the user activities. This parallel processing capability suggests that Specifier has the characteristics of a symbiotic DSS [Manh88].

Controller has functionality similar to executive in Requirements Apprentice architecture [Reub91] as it also provides high-level control of the reasoning performed. Specifier however controls reasoning of various components compared to a single component (Cake) in Requirements Apprentice.

**Statement analyzer:** The statement analyzer analyses user statements and formalizes it for further action. It works with a limited vocabulary, uses dictionaries of synonyms,
nouns, and verbs, and uses key words for analyzing domain information. It draws inferences using rules and pattern matching, and confirms its understanding by paraphrasing the statement.

**View Manager:** The view manager works on the model of the system currently being developed in the working memory. It acts on the analyzed input from the statement analyzer. It intelligently manages the view construction and keeps track of the portions of the view that are part of the organization model. Additionally, it also tracks portions of the view that require to be added to the organization model, and portions that are part of the private KB and model. For any attribute or service or message or association required to be added for an object, the view manager checks whether it is inherited and whether it is semantically valid.

**Model validator** checks for the consistency of the objects and specifications, and for completeness of the specifications. For example, if a message is used for getting service from another object, validator will check whether the service is defined, and whether required information for the service to perform is in the message or not. Checks validator performs includes checking whether model is semantically valid, whether all attributes of associations, attributes, inheritance, whole-part, methods and states are defined/present, whether specification of the method is complete/internally consistent.
**KB browser:** The KB browser enables querying and browsing through various knowledge-bases including IS knowledge-base. Its function is mainly information retrieval and presentation.

![Figure 11: Architecture for the Specifier](image-url)
Advisor advises users at different levels of abstraction using knowledge from various KBs and assists users in making decisions. For example, it may provide advise on possible group processes an user may like to use for designing a GDSS, or models the user may like to explore for designing a DSS in a domain.

Agenda manager: The agenda manager maintains a dynamic list of all pending issues identified by the other components in the specifier, suggests the order of addressing them, but enables users to select and work on them in a random order. It acts like a 'blackboard' for posting problems identified during validation and evaluation of the system being worked on. System specifications will not be acceptable to the controller if agenda has any pending issue.

Modifications manager: The modification manager provides required services for incorporating changes or modifications in the IS. For example, if any change is to be effected, it identifies the implications of the modification on the information systems in place in the organization; validates the modification and propagates the changes through all systems.

Default designer: The default designer provides the facility to refines and instantiates the higher level abstract objects to the lowest level of abstraction using designer KB that specifies the default design criteria of the organization. Default designing of the system can be used for implementing an IS quickly.
Presenter prints the documentation for the system. It includes complete object and task specifications, scenarios generation, state transition diagrams, and analysis of the usage of different objects.

The central idea of the architecture of specifier revolves around reuse of existing software components. It provides necessary tools and controls to enable composition based approach towards IS development [Bigg89]. The approach adopted here differs from other reuse based composition based approaches like one taken in ITHACA as in our approach search for reusable components is preference based and from multiple sources. ISKB is most preferred as it contains components already adapted to organization needs and has high likelihood for reuse without modification. Components from DKB are preferred next.

4. Organizational Knowledge Base

OKB can be organized as a set of different types of organization specific knowledge bases. These together act as the repository of the current status of the knowledge accumulated over a period. Models KB, policies and procedures KB, external environment KB, design KB, and private KB are all part of an OKB.

The models KB has all the models used within the organization. These models may be decision models, analysis models, interaction models, or any other abstract models with defined interaction and behavior with other organizational
entities. These models may be organized in a hierarchical fashion with organization-specific models at a higher level and more specific instantiations at the lower level. For example, an organization with multiple plants may have a generic organization-specific production model, and its instantiations as specific models for the individual plants.

A **private KB** acts as a repository for processing/models/decision objects in the private domains of the individuals required for satisfying their individual needs and support their individual working styles. Specifications for a private IS will be a view of the union of the organization model and the private KB. The private KB may include user specific instantiations of the objects in the organization model.

The **systems environment KB** has the knowledge about hardware, software, communication tools, presentation tools, and facilities in the organization. Besides instantiation of design objects and transformation of specification language into executable code, it assists in planning and integration of the target environments, and smooth migration to a new environment.

The **designer KB** contains knowledge about the IS design policies and guidelines the organization follows. For example, it may specify that all user interactions must support 'windows' environment. This knowledge is used to facilitate the design refinements and default design generation from the available specifications for a system.
The **external environment KB** has knowledge about entities external to the organization, along with implications for the organization. It includes the knowledge about competitors, suppliers, customers, governments, economic and business environments.

Any knowledge-base that is specific to an organization is a candidate to be part of its KB. KB that need to be included in OKB depends on individual characteristics of an organization, types of systems it has and it plans to develop, cost of developing that KB, and the extent to which the organization wants to ensure that the knowledge is considered during systems development.

5. **Domain Knowledge Base**

The domain KB is a collection of knowledge-bases of various domains. These domains can be functional domains like accounting, finance, manufacturing, etc. We include in the scope of DKB any knowledge-base that has generic knowledge about certain area. Thus DKB includes models knowledge-base, decision theory and decision support systems knowledge base, design knowledge-base, etc.

The **models KB** consists of generic models like statistical models, management science models like LP models, and business models like finance models with knowledge to assist the user in applying and using them effectively. It may include meta-models instead of models, and generate a model for specifier when needed. For example, it can generate
an LP model with required number of variables and make it available to the specifier.

The design KB has logically designed generic objects (e.g., windows, reports) with lowest level instantiations and all services completely specified in a specification language. Typical design objects included in this KB includes objects to manage user interface (including presentation, presentation manipulation methods and dialogue management).

Using knowledge-based management system for managing domain knowledge has not been used in many knowledge-based approaches for systems development. Many systems have usually taken approach similar to on-line libraries of subroutines. Requirements Apprentices uses cliché library as a declarative repository of commonly occurring structures [Reub91]. IDeA contains library of design schemas and of design refinements [Luba89]. It is based on data-flow representation. TEMPORA stores information in a relational database management system, SYBASE [Louc92]. We prefer a knowledge-based management system approach to keep specifier less complex by off-loading knowledge-base management functions, and keep DKB independent and portable.

6. Systems Development Knowledge Base

System Development Knowledge base (SKB) is another organization independent knowledge base in the OSD environment to support system development function. It has the necessary knowledge to validate and ensure consistency of
the view of the IS being worked upon, including all the lower level instantiations of higher level objects. It also has the knowledge for performing transformations of specifications into executable code, a dictionary of synonyms to avoid proliferation of objects, and the knowledge to support users during all the phases of system development and implementation. This KB is extensible.

SKB is organized as collection of various systems development related technical knowledge. Some knowledge bases it has are as follows. **Systems modeling knowledge base** consists of knowledge related to modeling the real world application. It includes knowledge related to presentation of the model in terms of charts etc., preparing specification document, and performing syntactic validation of the specifications. This knowledge enables representing specifications as, say, data-flow diagrams, from the underlying representation OSD uses (object-oriented).

**Transformation knowledge-base** has knowledge for transforming high level specifications into code, constrained by the technical environment in which the application will operate. Knowledge about features and constraints of various technical environment is in **platforms knowledge-base**. Thus if Microsoft C is targeted coding language on an IBM compatible PC, the knowledge-base will provide knowledge about features supported by this implementation of C.

Knowledge about generating designs from functional
specifications, and evaluating various design alternatives is in design knowledge-base.

7. IS knowledge Base (ISKB)

ISKB implements the organization model. It stores and manages all software components of the organizations. It maintains view definitions of the applications. Tools in the OSD process interact primarily by manipulating information stores or is a candidate for storing in ISKB.

Application software components stores in ISKB represent the complete knowledge of analysis, design and implementation accumulated in the organization over a period of time. They are stored as objects, and internally have object-oriented representation. These objects are structured into following main categories.

Analysis objects: objects that result from requirements analysis of applications.

Task objects: objects that describe sequence of operations on application objects using object methods to achieve the task objective. Collection of various tasks achieve application objective.

Generic design objects: objects that organization uses for representing design of analysis objects (i.e., objects like windows, reports, user-interface, etc.).

Implementation objects: Objects that correspond to actual implementation of analysis and design objects in the organization.
Application design objects: Objects that describe instantiation of analysis objects using design objects and relate these to the implementation objects.

View objects: Objects that describe views of various applications in the organization.

8. Summary

In this chapter we presented an architecture for implementing the OSD model. We also discussed the components of the SPECIFIER, which implements the specification development process.

The architecture we presented is knowledge-based and has the benefits of a knowledge-based management system. Various knowledge-base create the organizational systems development environment that support complete systems development process using the Specifier and a Synthesizer. The proposed architecture supports export and import of knowledge not specific to an organization, off-loads the knowledge management functions from the Specifier and the Synthesizer to the various knowledge-bases, and supports high reusability.
CHAPTER VI
SPECIFIER

1. Introduction

In this chapter we describe our research prototype, Specifier, that supports the organization model (OM) based information systems (IS) development. The Specifier assists users in OM-based IS development. It supports object-oriented development, manages the OM, and the view of the application on which a user is working. It stores specifications of all systems expressed in object-oriented terms into one organizational model. During the development process, it enforces reuse, avoids redundancy, validates specifications under development, validates OM before accepting enhancements, and maintains the list of pending issues to be addressed before specifications and enhancements are accepted. It uses forward chaining to assert new facts and dependency directed reasoning to retract deductions automatically.

Resources used for developing the Specifier are relatively modest compared to some other knowledge-based software development systems. "ITHACA is a 5-year, 100 person-year/year" project [Fugi92]. Programmer's Apprentice [Rich88] is an ongoing project for several years at MIT.
Specifier is developed using Lisp and an expert system shell, Goldworks, from Gold Hill Computers Inc., on a 486 based IBM PC compatible machine.

2. The Organization Model Structure

As discussed before, the OM integrates and stores complete knowledge about all specifications in the organization. In this chapter, we use OM and ISKB interchangeably. ISKB is information systems knowledge-base and is an implementation of the OM (refer chapter V).

The OM meta-model discussed in chapter IV for storing specifications in object-oriented representation is implemented using a lattice of frames.\(^1\) Figure 12 shows the hierarchy among the frames. ORG-MODEL is the root frame for the OM. OM-OBJECTS, OM-ATTRIBUTES, OM-SERVICES, TASK-SPECS, and OM-RELATIONS are direct descendants of the ORG-MODEL. OM-OBJECTS is a class for various objects in the OM. OM-ATTRIBUTES is a class for objects that describe attributes of objects in OM-OBJECT. OM-SERVICES objects describe services or methods associated with objects of OM-OBJECTS. OM-RELATIONS objects describe relations among objects of OM-OBJECTS. OM-RELATIONS has three specializations: PART-OF-RELN, INSTANCE-RELN, and GENSPEC-RELN.

\(^1\)Goldworks uses frame as a template or schema for some class of objects; it corresponds to a class in object-oriented terminology. Lattice consists of frames connected in a hierarchy that supports direct inheritance.
Figure 12: Lattice of OM-related frames in Specifier
Frames have slots, which correspond to attributes in object-oriented terminology. Following define-frame statement\(^2\) gives the slots associated with the ORG-MODEL frame.

```
(DEFINE-FRAME ORG-MODEL
  (APPLN :DOC-STRING "If an application uses this object, its name is here." :PRINT-NAME "Used-in-Applications" :EXPLANATION-STRING "applications using this object" :MULTIVALUED T :CONSTRAINTS (:INSTANCE-OF APPLICATIONS))
  (NAME)
  (POLICY :MULTIVALUED T :CONSTRAINTS (:INSTANCE-OF POLICIES))
  (CONSTRAINTS :MULTIVALUED T :CONSTRAINTS NIL)
  (SYNONYMS :MULTIVALUED T)
  (POLICY-MOD :MULTIVALUED T :CONSTRAINTS (:INSTANCE-OF POLICIES)))
```

Slots in ORG-MODEL are inherited by every object in the OM. Hence, all objects have a slot for APPLN, which is multi-valued and an instance of APPLICATIONS. It records the applications which has this object in its view. Also, all objects in OM have POLICY slot. It is used for linking the OM objects with policies in the policies KB.

SYNONYMS slot is used for recording synonyms organization uses for the name of the object. Hence, if organization refers 'parts' and 'items' interchangeably, 'parts' will be in SYNONYMS if 'items' is used as the frame name. Information in this slot is used to avoid proliferation of objects and to decrease the communication gap between a user and an analyst.

---

\(^2\)These statements are used in GoldWorks for creating frames.
OM-OBJECT, in addition to the ORG-MODEL slots has slots given in the following define-frame statement:

(DEFINE-FRAME OM-OBJECT
 (:IS ORG-MODEL)
 (TYPE)
 (WHEN-ADDED)
 (WHEN-DELETED)
 (WHEN-MODIFIED)
 (WHEN-ACCESSED))

The type slot records the type of object. Various type of objects in the OM are discussed in chapter IV. Above definition shows that OM-OBJECT is a child of ORG-MODEL. The four WHEN slots are triggers for data-manipulation operation. They specify additional activities to be done whenever the object is added, deleted, modified, or accessed. Following is an instance of frame OM-OBJECT. It is used in two applications, OCS and PS.

(DEFINE-INSTANCE PRODUCT
 (:IS OM-OBJECT)
 (APPLN OCS PS)
 (SYNONYMS GOODS)
 (KEYWORDS ITEMS GOODS))

OM-OBJECT has methods, called handlers in GoldWorks, that provide necessary services like making a list of OM-ATTRIBUTES associated to the object; or making a list of only those OM-ATTRIBUTES that are present in the view; making list of parents, of children, etc. Appendix B gives the complete list of handlers for OM-OBJECT along with the definitions. These handlers are used for view and OM management.
**Figure 13: Instance of an Application frame.**
Instances of the APPLICATIONS frame are the applications in the OM. Figure 13 is an instance of the APPLICATION frame. Appendix A includes the frame definition for APPLICATION and for other frames in the OM.

3. Knowledge as Rules and Assertions

Along with organizing the knowledge as frame-based structure, knowledge in the Specifier is organized as rules and assertions. Rules are used for making inferences and deductions. Hence they are used for recording active knowledge of the expert. Assertion is a factual knowledge. Rules use assertions to draw inferences. Hence an assertion, which represents a fact, is a passive knowledge [Gold89].

Rules have an antecedent part and a consequent part. The antecedent or IF part corresponds to a condition or a set of conditions. The consequent or THEN part corresponds to an action or a set of actions that is taken if the IF part is satisfied by available facts or knowledge. For example, consider a following rule from the Specifier knowledge-base³:

(DEFINE-RULE APL-OBJ-SERV1
 (:PRINT-NAME "serr-in-view-then-obj-in-view"
 :DOC-STRING "If serv of an obj is in a view, then the object is in view" :DEPENDENCY T)
 (INSTANCE ?SERV IS OM-SERVICES
 WITH OBJECT ?OBJ
 WITH APPLN ?APL)

(AND

³DEFINE-RULE is syntax for creating a rule in the expert system package used.)
This rule asserts the instance of OM-OBJECT in the view if its service is included in the view. The following rule, DEL-INST-CHK-APPLN does not allow an object to be deleted if it is used in a view.

(DEFINE-RULE DEL-INST-CHK-APPLN
 (:EXPLANATION-STRING "Do not delete object if any application is using it.")
 (INSTANCE D-MAINT-INST IS D-OBJECT
 WITH CUR-INST ?I
 WITH DELETE :YES)
 (INSTANCE ?I IS ? WITH APPLN ?A)
 THEN
 (SLOT-RETRACT-VALUE D-MAINT-INST DELETE)
 (EVALUATE
 (NOTIFY
 (STR "Cannot delete; used in APPLICATION" ?A)))))

Specifier has rules, among others, for creating a view, maintaining the view and the OM, validating views and the OM, searching the knowledge-base; creating, maintaining and retracting a pending issue. Appendix C lists some of the rules in the Specifier.

Assertion represents a factual or passive knowledge. Knowledge stored in the frames are facts and hence assertions. As these assertions are structured, they are referred as structured assertions. Unstructured assertions refer to facts entered in the knowledge-base without using any structure. Consider the following examples of unstructured assertions:

1. (OM-ADENDA OM-OBJECT SUPPLIER HAS-NO-ATTRIBUTES)
First assertion is for the fact that SUPPLIER object has no attributes defined for it. Second assertion states that current application on which user is working on is OCS.

4. Reasoning Mechanisms

Specifier uses forward chaining for making inferences from available facts, or assertions. Forward chaining is reasoning from facts to the conclusions asserted from those facts [Giar89]. In Specifier, patterns in the antecedents of the rules are matched against the assertions in the knowledge-base. Rules whose patterns are satisfied are instantiated and placed on the agenda for firing. Rules with the highest priority are executed first. Consider the following DEL-INST rule.

```
DEFINE-RULE DEL-INST
( :EXPLANATION-STRING "Deletes the instance if no rule of higher priority retracts :yes from delete slot of d-maint-inst."
 :PRIORITY -1000)
(INSTANCE D-MAINT-INST IS D-OBJECT WITH DELETE :YES WITH CUR-INST ?I)
THEN
(DELETE-INSTANCE ?I)
(EVALUATE (NOTIFY (STR ?I 'IS 'DELETED)))
(SLOT-RETRACT-VALUE D-MAINT-INST DELETE)
(SLOT-RETRACT-VALUE D-MAINT-INST CUR-INST))
```

This rule gets activated when user initiates a delete-instance request. As the priority of this rule is -1000 (lowest), it gets fired only if all other rules that get activated have fired first. If any of these rules retract the request for delete, DEL-INST is not fired. One such rule
which retracts a delete request is DEL-INST-CHK-APPL discussed before.

New assertions in the knowledge-base are made as a result of firing of the rules. Some assertions are dependent on other assertions. For example, if a service of an object is used in the view, that object is asserted in the view by rule APL-OBJ-SERV1 presented before. If later the service is dropped from the view, and it was the only cause for that instance of OM-OBJECT to be in the view, this assertion needs to be retracted. Specifier retracts such dependent assertions by maintaining dependency relationship between assertions, and retracting the assertion if none of the assertions on which it depends is in the knowledge-base. Dependency relationship is maintained for rules with dependency clause set to true (see APL-OBJ-SERV1 above).

5. View Creation, Generation, Management, and Validation

Specifier creates, manages and validates a view using the organization model, knowledge related to these in the rule-base, inference engine of the expert system shell, and various programs written in a LISP language.

Generate-view algorithm is implemented using LISP language. It executes and loads the generated view in the working-memory when user selects an application for working. It generates a view by checking every instance of OM-OBJECT for being used in the current application. For every instance used, it checks whether instance of OM-ATTRIBUTES,
OM SERVICES, RELATIONS are used by the application. It also checks instances of TASK-SPECS. All instances used in the current application are loaded in the working memory.

View enhancement is controlled and reuse is enforced by combinations of rules, dependency-directed reasoning and LISP code. Any enhancement required to the view is first searched in the OM. Current system performs search using keywords and synonyms. Following rule AI-KW-OM searches the OM for objects with matching keywords, and presents them for selection to the user:

```
(DEFINE-RULE AI-KW-OM ()
  (INSTANCE ?I IS D-OBJECT WITH ADD :YES
           WITH CUR-INST ?C
           WITH FRAME ?F
           WITH KEYWORDS ?K)
  (UNKNOWN (INSTANCE ?C IS ORG-MODEL))
  THEN
  (EVALUATE (MAKE-ATTEMPT (LIST 'INSTANCE '?X 'IS ?F
                             'WITH 'KEYWORDS ?K
                             'WITH-UNKNOWN 'APPLN (CURRENT-APPLN)) :NAME 'X))
  (EVALUATE (LET ((Y (QUERY-ALL 'X :THEN '((EVALUATE ?X)))) (X NIL)) 'INSTRUCTIONS)
           (STR "View enhancement: FOLLOWING OBJECTS HAVE MATCHING KEYWORD" ?K "SELECT X TO SEE DETAILS")
           (IF Y (UPDATE-ELEMENTS 'POP-T-OUTPUT Y 'MAINT-INST-I) (FORWARD-CHAIN))))
```

Some other rules related to enhancement are AI-IN-APPL, AI-IN-OM, ADD-FAIL-INCL, ADD-EFFECT, INCLUDE-APPL-FAIL, INCLUDE-APPL (refer appendix C). In some cases, view-enhancement is automatically performed by the system, as in APL_OBJ-SERVI above.

Validation of views and the OM is performed using validation rules in the rule-base. Validation process is
triggered by asserting (VERIFY VIEW) or (VERIFY OM) in the working memory. Validations Specifier performs includes checking for internal consistencies, enforcing key-words, checking for linkages with policies. Following rule creates a 'pending-issue' assertion for a service in the view:

```
(DEFINE-RULE VIEW-SERVICE-NO-POLICY
 (:DEPENDENCY T)
 (VERIFY VIEW)
 (CUR-APPLN ?A)
 (INSTANCE ?S IS OM-SERVICES
  WITH APPLN ?A
  WITH-UNKNOWN POLICY ?)
 THEN
 (AGENDA OM-SERVICES ?S NOT-LINKED-TO-POLICY))
```

Figures 14 and 15 show some pending issues Specifier identifies and displays for a view and the OM.

6. User Interface and Interaction with the Specifier

Specifier operates in the windows environment. User interface is menu-driven. Main menu in the current system is a horizontal bar with menu options at the top of the window screen (see figure 16). Pull-down menus are attached to them.

User interaction is explained with the help of various figures generated during the development of order confirmation system (OCS) for a small organization that makes custom-order items. This hypothetical organization gives quotations for the enquiry received, and on receiving the order, confirms the acceptance of the order if it finds the same acceptable.
An user typically starts the interaction by selecting the 'Application' option. If the user selects 'Existing' from the pull-down menu, a pick-list of existing applications is displayed for selection (figure 17).

![View - pending issues](image)

Figure 14: View - pending issues.
Organizational Model pending issues.

Figure 15: Organizational Model pending issues.
Figure 16: Specifier screen with menu bar
Figure 17: Pick-list of existing applications in the OM.
On making a selection, view of the selected application is generated and a window ('Application View') showing objects in the view is displayed. Name of the selected or a newly created application is displayed as the window name after 'Specifier' (figure 16).

'Application View' object has various panels to display names of instances of different types of objects in the metamodel. Each instance has an 'X' button associated with it, which opens the instance when clicked upon.

If button for ENQUIRY is clicked, details of the ENQUIRY object as in the view is displayed in a new window called 'Application Model Object'. Figure 18 shows the window displayed. It gives information like attributes, services, parents, children of ENQUIRY, relations in which ENQUIRY participates, other objects it contains, etc.. 'X' button against an object name on this window opens the object for further operation. Figure 19 shows the window that is displayed if button against ENQ-QUO is selected. Buttons next to 'Attributes' and 'Services' initiate view-enhancement process for adding a new attribute or service.

Name of the instance of OM-OBJECT in 'Application Model Object' window is a button. If clicked, it opens a window that displays characteristics associated with the object and provides further options (figure 20).
Figure 18: Application object ENQUIRY
**Figure 19: ENQ-QUO** the application view
Figure 20: OM-OBJECT: ENQUIRY window
User can explore the organization model using 'Org-model' option on the horizontal bar. User can further specify the type of objects to be explored by making a selection from the pull-down menu (figure 21). If 'Objects' is selected, list of all objects in the OM are displayed as pick-list. Object selected from the pick-list displays complete information about the object in the OM (Figure 22). 'Organization Model Object' in figure 22 is similar to 'Application Model Object' except that one shows the information from the OM, other from the view.

User can similarly explore the view by using another option on the horizontal bar - 'View'. Selecting this option pulls down a menu for further selection (figure 23). 'Enhance-View' option (figure 24) on the menu bar gives various options for enhancing the view. It includes options for exploring and for quick ways of selecting objects from the OM. User uses 'Add' option to initiate the enhancement of the model directly. A pop-up screen asks the user to enter basic information like object type, name, keywords (figure 25). After the information is entered, the inference mechanism processes the option by checking against the view and the OM.
Figure 21: Specifier: Options for Org-model
Figure 22: Organization model object CUSTOMER
Figure 23: Specifier: View options
Figure 24: Specifier: Enhance-view options
Figure 25:Specifier: Enhance model screens

Figure 26:Specifier: Select objects from organization model window.
Figure 27: Specifier: Reuse analysis for a software component type.
Figure 28: Specifier: Validate options
User can avoid checking against the OM and include objects from the OM fast by using other four options from the 'Enhance-View'. On selecting an option, a pop-up screen displays the list of available objects in the OM for the user to select. Figure 26 shows a pop-up screen for the user to select objects of type OM-OBJECT.

User initiates validation of the view or the OM and checks the items for which action need to be taken by choosing 'Validate' option from the horizontal bar menu (figure 28). 'Validate-view' and 'Validate-OM' initiates validation process by asserting (VERIFY VIEW) and (VERIFY OM). 'Pending view-agenda' and 'Pending OM-agenda' display list of pending items (figures 14, 15).

'Reuse analysis' displays the usage of various software components in different application. Currently, two options are provided. One option is to explore reuse between any two specified application. Another option displays reuse of various instances of a particular component type in different applications. Figure 27 shows the two options and displays reuse analysis for a component type. User can use this for analyzing the impact of modification.

7. Conclusions

In this chapter we discussed our current implementation of the Specifier and have demonstrated the feasibility of our key ideas, the concept of the organization model and of
views. Our current version demonstrates that it is possible to create a view for an application from the OM, that specifications for an application need not be stored separately, and that reuse of software components developed in the organization is possible. We demonstrated that during the development process, our approach enforces reuse, avoids redundancy, validates specifications under development, validates OM before accepting enhancements, and maintains the list of pending issues to be addressed before specifications and enhancements are accepted.
CHAPTER VII
SUMMARY AND FUTURE RESEARCH DIRECTIONS

Information systems (IS) development and maintenance for the organizations today is a complex task. Given the competitive environment in which organizations operate today, they typically have or require systems that are integrated, internally consistent, and consistent with their latest operational and other business practices. As the dependence on the IS is high, changes in requirements or needs have to be effected quickly and correctly.

Along with technological innovation, complexity of the IS in the organizations is increasing continuously. Thus we can expect that IS development and maintenance problems will increase further. Highly complex requirements, bounded rationality problem for developers, and high personal turnover makes it imperative that IS development and maintenance is supported as much as possible and with an organizational perspective.

We presented the Organizational Systems Development (OSD) model and the organization model based systems development method to intelligently support systems development and maintenance in a knowledge based environment with an organizational perspective. Besides the domain
knowledge and the systems development knowledge, we integrated the support from the organizational knowledge and the organizational model in the OSD environment to enable system development that is highly reuse based, considers other IS in the organization, and inherently supports integration.

We also presented the concepts of the organizational model as integration of all system specifications in the organization, and application views as a minimal subset of the organization model that specifies the application system completely. We used these concepts for the organization model based IS development and showed its advantages like high reuse, high productivity, capture of design knowledge, specification based maintenance, consistent implementation of changes across IS, etc..

To maximize the benefits of the organization model, we developed two criteria for evaluating the representation with the objective of maximizing reuse, integration and minimizing effort. We showed that to achieve these objective, representation has to be such that the maximum knowledge about the real world is captured in the model for the real world, and task specifications to achieve the objective of the application should have minimal knowledge content.

Based on the criteria developed, we selected the object oriented representation. For an object oriented representation of the specification in the organization model (OM), we developed a meta-model for the OM. We classified the
objects in the OM as analysis objects, application design objects, generic design objects, task specs objects, and implementation objects; and discussed the steps and processes in the OM-based IS development.

For implementing the OSD model, we chose knowledge-based systems for storing and maintaining various knowledge as a logical choice. We presented the architecture for the OSD model and for the Specifier. This architecture supports import and export of the knowledge that is not specific to the organization.

We used an expert system shell and implemented the key components of the Specifier in a windows environment that demonstrated the feasibility of OM based approach. We implemented various algorithms for view and OM management, and demonstrated the view generation from the OM, reuse, localizing changes required due to changes in any component, controlled enhancement of OM, and validation of the view and of the OM. We used rules and assertions for capturing knowledge, and forward chaining with dependence maintenance as a reasoning mechanism.

Further research need to be done before the OSD model and OM-based IS development approach can be made practical for the organizations. Below we present some potential areas.

Application specification as a view of the organization model concept is similar to that of views in the relational database. With the basic feasibility demonstrated, a formal theory need to be developed and all operators need to be
formalized. A proper formal theory should provide a strong foundation for implementing OM-based IS development.

Efficient and effective search mechanism for searching a software component in the organizational model when enhancing the view is essential for the OM-based approach to be practical. Search should be efficient in terms of using the computing resources; it should be effective in terms of being able to retrieve a component if it is present in the OM. Synonym-based and keyword search may be unsuitable for a large OM. Research for identifying proper search mechanisms for the OM need to be done.

Different knowledge stored in the knowledge-based systems is accessed by the Specifier for use during IS development. To support import and export of knowledge, these knowledge-based systems need to be independent of the OSD process. Besides the search issues similar to that for an organization model discussed above, additional issue regarding interface protocol, representation of knowledge in terms of facts and reasoning mechanisms need to be further addressed in depth.

Various implementation issues need to be addressed before the OSD model is practical. Issues related to implementing the OSD model with the organizational knowledge and the OM not in place, implementing in a multi-project, multi-site, multi-personnel environment, capturing the knowledge of various legacy systems in the OM, developing and
providing supporting tools for using the OSD environment effectively, need to be studied further and addressed.

Field studies to study the effect of the OSD environment on the IS development and maintenance effectiveness, reuse, integration, and effect on the developers, need to be performed. These field study may give some critical insights that need to be considered for improving the OSD environment.

Our research presented here is a step towards enabling the organizations to develop and maintain IS with an organizational perspective. We expect that our work should trigger a stream of further research in the areas of IS development from an organizational perspective and knowledge-based support for IS development.
APPENDIX A

FRAMES IN THE SPECIFIER

(DEFINE-FRAME OSDE

() (DESCRIPTION) (KEYWORDS :MULTIVALUED T))

(DEFINE-FRAME ORG-MODEL

(:PRINT-NAME "Organization-model" :DOC-STRING "Parent of all org-model objects" :IS OSDE)
(APPLN :DOC-STRING "If an application uses this object, its name is here." :PRINT-NAME "Used-in-Applications" :EXPLANATION-STRING "applications using this object" :MULTIVALUED T :CONSTRAINTS (:INSTANCE-OF APPLICATIONS))
(NAME)
(POLICY :MULTIVALUED T :CONSTRAINTS (:INSTANCE-OF POLICIES))
(CONSTRAINTS :MULTIVALUED T :CONSTRAINTS NIL)
(SYNONYMS :MULTIVALUED T)
(POLICY-MOD :MULTIVALUED T :CONSTRAINTS (:INSTANCE-OF POLICIES)))

(DEFINE-FRAME OM-ATTRIBUTES

(:IS ORG-MODEL)
(DEFAULT)
(ONE-OF)
(RANGE)
(TYPE)
(WHEN-ADDED :CONSTRAINTS (:LISP-TYPE FUNCTION)) (WHEN-MODIFIED :CONSTRAINTS (:LISP-TYPE FUNCTION)) (WHEN-ACCESSED :CONSTRAINTS (:LISP-TYPE FUNCTION)) (WHEN-DELETED :CONSTRAINTS (:LISP-TYPE FUNCTION))
(OBJECT :CONSTRAINTS (:INSTANCE-OF OM-OBJECT)

:DOC-STRING "name of the object having this attribute." :PRINT-NAME "Attribute-of-object"))
(DEFINE-FRAME OM-OBJECT
  (:IS ORG-MODEL)
  (POLICY :MULTIVALUED T)
  (TYPE)
  (WHEN-ADDED)
  (WHEN-DELETED)
  (WHEN-MODIFIED)
  (WHEN-ACCESSED))

(DEFINE-FRAME OM-RELATIONS
  (:IS ORG-MODEL)
  (OBJECT1 :CONSTRAINTS (:INSTANCE-OF OM-OBJECT))
  (OBJECT2 :CONSTRAINTS (:INSTANCE-OF OM-OBJECT))
  (MAX-2FOR1)
  (MIN-2FOR1)
  (MAX-1FOR2)
  (MIN-1FOR2))

(DEFINE-FRAME INSTANCE-RELN
  (:IS OM-RELATIONS)
  (MAX-1FOR2 :CONSTRAINTS (:RANGE (0 100)))
  (CONSTRAINTS :CONSTRAINTS (:LISP-TYPE FUNCTION))
  (RELN-1TO2)
  (RELN-2TO1)
  (WHEN-ADDED)
  (WHEN-MODIFIED)
  (WHEN-DELETED)
  (WHEN-ACCESSED))

(DEFINE-FRAME GENSPEC-RELN
  (:DOC-STRING "object1 is parent, object2 is child"
    :IS OM-RELATIONS))

(DEFINE-FRAME PART-OF-RELN
  (:DOC-STRING "Object1 is part of Object2"
    :IS OM-RELATIONS)
  (WHEN-ADDED)
  (WHEN-DROPPED)
  (WHEN-ACCESSED))

(DEFINE-FRAME OM-SERVICES
  (:IS ORG-MODEL)
  (CONSTRAINTS :CONSTRAINTS (:LISP-TYPE FUNCTION))
  (PRECONDITION)
(POST-CONDITIONS :CONSTRAINTS (:LISP-TYPE FUNCTION)) (RESULT)
(PARAMETERS :CONSTRAINTS (:LISP-TYPE LIST)) (ACTION-PROCESS :CONSTRAINTS
(:LISP-TYPE FUNCTION)) (OBJECT
 :CONSTRAINTS (:INSTANCE-OF OM-OBJECT)))

(DEFINE-FRAME TASK-SPECS
  (:IS ORG-MODEL)
  (PRECONDITION)
  (FREQUENCY)
  (INITIATED-BY)
  (PREDICATE-ARG)
  (PREDICATE)
  (STEPS-LISTS :MULTIVALUED T
 :CONSTRAINTS (:LISP-TYPE LIST)))

(DEFINE-FRAME APPLICATIONS
  (:IS OSDE)
  (NAME)
  (OBJECTS :CONSTRAINTS (:INSTANCE-OF OM-OBJECT) :MULTIVALUED T)
  (RELATIONS :CONSTRAINTS (:INSTANCE-OF OM-RELATIONS) :MULTIVALUED T)
  (TASKS :CONSTRAINTS (:INSTANCE-OF TASK-SPECS) :MULTIVALUED T)
  (CUR-OBJECT :NO-SAVE T
 :CONSTRAINTS (:INSTANCE-OF OM-OBJECT))
  (CUR-OBJ-SERV :CONSTRAINTS (:INSTANCE-OF OM-SERVICES) :MULTIVALUED T)
  (CUR-OBJ- PARENTS :CONSTRAINTS (:INSTANCE-OF OM-OBJECT)
 :MULTIVALUED T)
  (CUR-OBJ-INST-RELS :CONSTRAINTS (:INSTANCE-OF INSTANCE-RELN)
 :MULTIVALUED T)
  (CUR-OBJ-PART-OF :CONSTRAINTS (:INSTANCE-OF OM-OBJECT)
 :MULTIVALUED T)
  (PARTS-OF-CUR-OBJ :CONSTRAINTS (:INSTANCE-OF OM-OBJECT)
 :MULTIVALUED T)
  (CUR-OBJ-ATRBS :MULTIVALUED T
 :CONSTRAINTS (:INSTANCE-OF OM-ATTRIBUTES))
  (POLICIES :CONSTRAINTS (:INSTANCE-OF POLICIES)
 :MULTIVALUED T)
  (SUBJECTS :MULTIVALUED T))

(DEFINE-FRAME ORG-KB
  (:IS OSDE))
(DEFINE-FRAME POLICIES
  (:IS ORG-KB)
  (REFERENCE)
  (PARENT-POLICY)
  (SUBJECTS)
  (OBJECTS)
  (GOALS)
  (CONSTRAINTS)
  (TYPE)
  (CHILD-POLICIES)
  (CHILD-P-ORGANIZATION)
  (POLICY-MOD :CONSTRAINTS (:ONE-OF (YES NO))))

(DEFINE-FRAME SYNONYM-KB
  (:IS OSDE)
  (OBJECT :CONSTRAINTS (:LISP-TYPE SIMPLE-STRING))
  (SYNONYMS :CONSTRAINTS (:LISP-TYPE LIST)))

(DEFINE-FRAME DOMAIN-KB
  (:IS OSDE))

(DEFINE-FRAME D-OBJECT
  ()
  (SLOT1)
  (SLOT2)
  (SLOT3)
  (TASKS)
  (CUR-INST)
  (ADD)
  (DELETE)
  (MODIFY)
  (REMOVE :CONSTRAINTS (:ONE-OF (:YES :NO :ATTEMPT)))
  (RENAME)
  (COPY)
  (VALIDATE)
  (KEYWORDS :DEFAULT-VALUES NIL
             :MULTIVALUED T)
  (FRAME :CONSTRAINTS (:ONE-OF
                       (OM-OBJECT OM-ATTRIBUTES OM-SERVICES TASK-SPECS GENSPEC-
                       RELN INSTANCE-RELN PART-OF-RELN)))
  (SUBJECTS :MULTIVALUED T)
  (INCLUDE)
  (REUSE)
  (APPLN :CONSTRAINTS (:INSTANCE-OF APPLICATIONS
                       :MULTIVALUED T)
          (OBJECT)
          (RELN))
(REUSE-FRAME :CONSTRAINTS (:ONE-OF (OM-OBJECT GENSPEC-RELN INSTANCE-RELN PART-OF-RELN)))

(DEFINE-FRAME D-OM-OBJECT
  ()
  (CHILDREN :CONSTRAINTS (:INSTANCE-OF OM-OBJECT)
    :MULTIVALUED T)
  (INST-RELNS :CONSTRAINTS (:INSTANCE-OF INSTANCE-RELN)
    :MULTIVALUED T)
  (PART-OF :CONSTRAINTS (:INSTANCE-OF OM-OBJECT)
    :MULTIVALUED T)
  (CONTAINS :MULTIVALUED T
    :CONSTRAINTS (:INSTANCE-OF OM-OBJECT))
  (PARENTS :MULTIVALUED T
    :CONSTRAINTS (:INSTANCE-OF OM-OBJECT))
  (ATTRIBUTES :CONSTRAINTS (:INSTANCE-OF OM-ATTRIBUTES)
    :MULTIVALUED T)
  (SERVICES :CONSTRAINTS (:INSTANCE-OF OM-SERVICES)
    :MULTIVALUED T)
  (NEW-ATR)
  (NEW-SERVICE)
  (NEW-PARENT)
  (NEW-CHILD)
  (NEW-PARTOF)
  (NEW-CONTAINS)
  (NEW-RELN)
  (ALL-ATTRIBUTES)
  (ALL-PARENTS)
  (ALL-CHILDREN)
  (OBJ-SYMBOL)
  (CUR-APPLN))
APPENDIX B
HANDLERS IN THE SPECIFIER

;;;;;;;;;;;;;;;;;;;;;;;;;; OM-OBJECT handlers;;;;;;;;;;;;;;;;;;;;;;;;;;

(define-handler (om-object attributes) ()
(make-attempt
  `(instance ?atr is om-attributes with object
    ,(gw-name self))
  :name 'a-o-atr)
  (query-all 'a-o-atr :then '((evaluate ?atr))))
)

(define-handler (om-object view-attributes) ()
(make-attempt
  `(instance ?atr is om-attributes with object , (gw-name self)
    with appln , (current-appln))
  :name 'a-o-atr)
  (query-all 'a-o-atr :then '((evaluate ?atr))))
)

(define-handler (om-object services) ()
(make-attempt
  `(instance ?x is om-services with object , (gw-name self))
  :name 'ax)
  (query-all 'ax :then '((evaluate ?x))))
)

(define-handler (om-object view-services) ()
(make-attempt
  `(instance ?x is om-services with object , (gw-name self)
    with appln , (current-appln))
  :name 'ax)
  (query-all 'ax :then '((evaluate ?x))))
)

(define-handler (om-object parents) (&rest lists)
(make-attempt
  `(instance ?i is genspec-reln
    with object2 , (gw-name self)
    with object1 ?p)
  :name 'ax)
  (if (not (first lists)) (query-all 'ax
    :then '((evaluate ?p)))
  (query-all 'ax :then '((evaluate (list ?i ?p)))))
)
(define-handler (om-object children) (&rest lists)
  (make-attempt `(instance ?i is genspec-reln with object1
                   ,(gw-name self)
                   with object2 ?p)
                 :name 'ax)
  (if (not (first lists)) (query-all 'ax :then '((evaluate ?p))) (query-all 'ax :then '((evaluate (list ?i ?p)))))
)

(define-handler (om-object view-parents) (&rest lists)
  (make-attempt `(instance ?i is genspec-reln
                       with object2 , (gw-name self)
                       with object1 ?p
                       with appln , (current-appln))
                 :name 'ax)
  (if (not (first lists)) (query-all 'ax :then '((evaluate ?p))) (query-all 'ax :then '((evaluate (list ?i ?p)))))
)

(define-handler (om-object view-children) (&rest lists)
  (make-attempt `(instance ?i is genspec-reln
                       with object1 , (gw-name self)
                       with object2 ?p
                       with appln , (current-appln))
                 :name 'ax)
  (if (not (first lists)) (query-all 'ax :then '((evaluate ?p))) (query-all 'ax :then '((evaluate (list ?i ?p)))))
)

(define-handler (om-object partof) (&rest lists)
  (make-attempt `(instance ?i is part-of-reln with object1
                       ,(gw-name self)
                       with object2 ?p)
                 :name 'ax)
  (if (not (first lists)) (query-all 'ax :then '((evaluate ?p))) (query-all 'ax :then '((evaluate (list ?i ?p)))))
)

(define-handler (om-object contains) (&rest lists)
  (make-attempt `(instance ?i is part-of-reln with object2
                       ,(gw-name self)
                       with object1 ?p)
                 :name 'ax)
(if (not (first lists)) (query-all 'ax :then '((evaluate ?p)))
(query-all 'ax :then '((evaluate (list ?i ?p)))))
)

(define-handler (om-object view-partof) (&rest lists) (make-attempt`
(instance ?i is part-of-reln
with object1 ,(gw-name self)
with object2 ?p
with appln ,(current-appln))
:name 'ax)
(if (not (first lists)) (query-all 'ax :then '((evaluate ?p)))
(query-all 'ax :then '((evaluate (list ?i ?p)))))
)

(define-handler (om-object view-contains) (&rest lists)
(make-attempt`
(instance ?i is part-of-reln
with object2 , (gw-name self)
with object1 ?p
with appln ,(current-appln))
:name 'ax)
(if (not (first lists)) (query-all 'ax :then '((evaluate ?p))) (query-all 'ax :then '((evaluate (list ?i ?p)))))
)

(define-handler (om-object inst-relns) ()
(make-attempt`
(instance ?r is instance-reln with
object1 , (gw-name self)
with object2 ?)
:name 'ax)
(make-attempt`
(instance ?r is instance-reln with
object2 , (gw-name self)
with object1 ?)
:name 'ay)
(let ((x (query-all 'ax :then '((evaluate ?r)))) (y (query-all 'ay :then '((evaluate ?r)))) )
(concatenate 'list x y))
)

(define-handler (om-object view-inst-relns) () (make-attempt`
(instance ?r is instance-reln
with object1 , (gw-name self)
with appln ,(current-appln)
with object2 ?)
:name 'ax)
(make-attempt`
(instance ?r is instance-reln with
object2 , (gw-name self)
with object1 ?
with object2 ?)
with appln , (current-appln))
  :name 'ay)
  (let ((x (query-all 'ax :then '((evaluate ?r))) ) ) (y
  (query-all 'ay :then '((evaluate ?r))) ))
  (concatenate 'list x y)
  )
 )
(DEFINE-RULE ADD-EFFECT
   (:PRIORITY -1000)
   (INSTANCE ?I IS D-OBJECT
      WITH ADD :YES
      WITH CUR-INST ?C
      WITH FRAME ?F)
   (INSTANCE D-MAINT-INST IS D-OBJECT
      WITH-UNKNOWN INCLUDE :YES)
   (UNKNOWN
      (INSTANCE ?C IS ?))
   THEN
   (INSTANCE ?C IS ?F)
   (EVALUATE
      (SETF
       (SLOT-VALUE ?C
          'KEYWORDS)
       (SLOT-ALL-VALUES ?I
          'KEYWORDS)))
   (EVALUATE
      (SETF
       (SLOT-VALUE ?C
          'APPLN)
       (CURRENT-APPLN)))
   (DELETE-INSTANCE ?I)
   (EVALUATE
      (NOTIFY "Model enhancement:" :RETURN (STR "Adding" ?C "of type" ?F)))
   (EVALUATE
      (MAINT-INST-IRD
         'X ?C)))

(DEFINE-RULE ADD-FAIL-INCL
   (:PRIORITY 10)
   (INSTANCE ? IS D-OBJECT
      WITH INCLUDE :YES)
   (INSTANCE ?I IS D-OBJECT
      WITH ADD :YES)
   THEN
   (SLOT-RETRACT-VALUE ?I ADD))
(DEFINE-RULE ADD-KW-MUST
   ()
   (OR
     (INSTANCE ?A IS D-OBJECT WITH-UNKNOWN CUR-INST ? WITH ADD :YES)
     (INSTANCE ?A IS D-OBJECT WITH-UNKNOWN FRAME ?
       WITH ADD :YES)
     (INSTANCE ?A IS D-OBJECT
       WITH ADD :YES
       WITH-UNKNOWN KEYWORDS ?))
   THEN
   (EVALUATE
     (IF
       (INSTANCE-P ?A)
       (DELETE-INSTANCE ?A))
     (EVALUATE
       (NOTIFY
         (STR
           "Model enhancement: Keywords, type and name are must.")
         ':RETURN
         (STR "Add unsuccessful."))))
   (DEFINE-RULE AI-IN-APPL
     ()
     (INSTANCE ?I IS D-OBJECT
       WITH CUR-INST ?C
       WITH ADD :YES)
     (INSTANCE CURRENT-APPLN IS D-OBJECT
       WITH SLOT1 ?A)
     (INSTANCE ?C IS ORG-MODEL
       WITH APPLN ?A)
     THEN
     (EVALUATE
       (NOTIFY "Model enhancement:" :RETURN
         (STR ?C "already present in application" ?A " Add
         unsuccessful")))
     (EVALUATE
       (IF
         (INSTANCE-P ?I)
         (DELETE-INSTANCE ?I)))
   (DEFINE-RULE AI-IN-OM
     ()
     (INSTANCE ?I IS D-OBJECT
       WITH CUR-INST ?C
       WITH FRAME ?F
       WITH ADD :YES
       WITH KEYWORDS ?K)
     (INSTANCE CURRENT-APPLN IS D-OBJECT WITH SLOT1 ?A)
     (INSTANCE ?C IS ORG-MODEL WITH-UNKNOWN APPLN ?A)
     THEN
     (SLOT-RETRACT-VALUE ?I ADD)
(EVALUATE
  (NOTIFY "Model enhancement:" :RETURN
    (STR ?C
      "Present in the Org Model. Verify
      and include in the view.")
    (EVALUATE
      (MAINT-INST
        'X ?C
        'INCLUDE))
    (EVALUATE
      (IF
        (INSTANCE-P ?I)
        (DELETE-INSTANCE ?I))))
  (DEFINE-RULE AI-KW-OM
    ()
    (INSTANCE ?I IS D-OBJECT
      WITH ADD :YES
      WITH CUR-INST ?C
      WITH FRAME ?F
      WITH KEYWORDS ?K)
    (UNKNOWN
      (INSTANCE ?C IS ORG-MODEL))
    THEN
    (EVALUATE
      (MAKE-ATTEMPT
        (LIST
          'INSTANCE
          '?X
          'IS ?F
          'WITH
          'KEYWORDS ?K
          'WITH-UNKNOWN
          'APPLN
          (CURRENT-APPLN)) :NAME
          'X)
    )
    (EVALUATE
      (LET
        ((Y (QUERY-ALL
              'X :THEN
              '((EVALUATE ?X))))
         (X NIL))
        (SETF
          (SLOT-VALUE
            'POP-T-OUTPUT
            'INSTRUCTIONS)
          (STR "View enhancement: FOLLOWING OBJECTS HAVE MATCHING
                  KEYWORD" ?K "SELECT X TO SEE DETAILS")
          (IF Y
            (UPDATE-ELEMENTS
              'POP-T-OUTPUT Y
              'MAINT-INST-I))
          )
        ))
(DEFINE-RULE APL-COBJ-ATR1
(:PRINT-NAME "apl-cur-obj-atr-view"
 :DEPENDENCY T)
 (INSTANCE ?APL IS APPLICATIONS
  WITH CUR-OBJECT ?COBJ)
 (INSTANCE ?ATR IS OM-ATTRIBUTES
  WITH OBJECT ?COBJ
  WITH APPLN ?APL)

 THEN
 (INSTANCE ?APL IS APPLICATIONS
  WITH CUR-OBJ-ATTRS ?ATR))

(DEFINE-RULE APL-COBJ-ATR2
 (:PRINT-NAME "apl-cur-obj-atr-view-2")
 (INSTANCE ?APL IS APPLICATIONS
  WITH CUR-OBJECT ?COBJ
  WITH CUR-OBJ-ATTRS ?ATR)

 (OR
  (INSTANCE ?ATR IS OM-ATTRIBUTES WITH-UNKNOWN OBJECT ?COBJ)
   (INSTANCE ?ATR IS OM-ATTRIBUTES
    WITH OBJECT ?COBJ WITH-UNKNOWN APPLN ?APL)
  )

 THEN
 (SLOT-RETRACT-VALUE ?APL CUR-OBJ-ATTRS ?ATR))

(DEFINE-RULE APL-COBJ-INSREL1
 (:DEPENDENCY T)
 (INSTANCE ?APL IS APPLICATIONS
  WITH CUR-OBJECT ?COBJ
  WITH RELATIONS ?R)

 (OR
  (INSTANCE ?R IS INSTANCE-RELN
   WITH OBJECT2 ?COBJ)
   (INSTANCE ?R IS INSTANCE-RELN
   WITH OBJECT1 ?COBJ)
  )

 THEN
 (INSTANCE ?APL IS APPLICATIONS
  WITH CUR-OBJ-INST-RELNS ?R))

(DEFINE-RULE APL-COBJ-INSREL2
 ()
 (INSTANCE ?APL IS APPLICATIONS
  WITH CUR-OBJECT ?COBJ
  WITH CUR-OBJ-INST-RELNS ?R)

 (OR
  (AND
   (INSTANCE ?R IS INSTANCE-RELN WITH-UNKNOWN OBJECT2 ?COBJ)
   (INSTANCE ?R IS INSTANCE-RELN WITH-UNKNOWN OBJECT1 ?COBJ)
   (INSTANCE ?APL IS APPLICATIONS WITH-UNKNOWN RELATIONS ?R)
  )

 THEN
(DEFINE-RULE APL-COBJ-PARENT2 ()
(INSTANCE ?APL IS APPLICATIONS
  WITH CUR-OBJECT ?COBJ
  WITH CUR-OBJ-PARENTS ?P)
(INSTANCE ?G IS GENSPEC-RELN
  WITH OBJECT2 ?COBJ
  WITH-UNKNOWN OBJECT1 ?P)
THEN
(SLOT-RETRACT-VALUE ?APL CUR-OBJ-PARENTS ?P))

(DEFINE-RULE APL-COBJ-PARENTS (:DEPENDENCY T)
(INSTANCE ?APL IS APPLICATIONS
  WITH CUR-OBJECT ?COBJ)
(INSTANCE ?REL IS GENSPEC-RELN
  WITH OBJECT1 ?PARENT
  WITH OBJECT2 ?COBJ)
THEN
(INSTANCE ?APL IS APPLICATIONS
  WITH CUR-OBJ-PARENTS ?PARENT))

(DEFINE-RULE APL-COBJ-PARTOF1 (:DEPENDENCY T)
(INSTANCE ?APL IS APPLICATIONS
  WITH CUR-OBJECT ?COBJ
  WITH RELATIONS ?R)
(INSTANCE ?R IS PART-OF-RELN
  WITH OBJECT2 ?POFF
  WITH OBJECT1 ?COBJ)
THEN
(INSTANCE ?APL IS APPLICATIONS
  WITH CUR-OBJ-PART-OF ?POFF))

(DEFINE-RULE APL-COBJ-SERV1 (:PRINT-NAME "apl-cur-obj-serv-view"
  :DEPENDENCY T)
(INSTANCE ?SERV IS OM-SERVICES
  WITH OBJECT ?COBJ
  WITH APPLN ?APL)
(INSTANCE ?APL IS APPLICATIONS
  WITH CUR-OBJECT ?COBJ
  WITH-UNKNOWN CUR-OBJ-SERV ?SERV)
THEN
(INSTANCE ?APL IS APPLICATIONS
  WITH CUR-OBJ-SERV ?SERV))

(DEFINE-RULE APL-COBJ-SERV2 (:PRINT-NAME "apl-cur-obj-serv-view-2")
(INSTANCE ?APL IS APPLICATIONS
  WITH CUR-OBJECT ?COBJ
  WITH CUR-OBJ-SERV ?SERV)
(OR
(INSTANCE ?SERV IS OM-SERVICES
 WITH OBJECT ?OBJ WITH-UNKNOWN APPLN ?APL)
(INSTANCE ?SERV IS OM-SERVICES WITH-UNKNOWN OBJECT ?OBJ))
THEN
(SLOT-RETRACT-VALUE ?APL CUR-OBJ-SERV ?SERV))

(DEFINE-RULE APL-OBJ-ATR1
 (:DEPENDENCY T)
 (INSTANCE ?ATR IS OM-ATTRIBUTES
  WITH OBJECT ?OBJ
  WITH APPLN ?APL)
 (INSTANCE ?OBJ IS OM-OBJECT
  WITH-UNKNOWN APPLN ?APL)
THEN
(INSTANCE ?OBJ IS OM-OBJECT
 WITH APPLN ?APL))

(DEFINE-RULE APL-OBJ-SERV1
 (:PRINT-NAME "serr-in-view-then-obj-in-view"
 :DOC-STRING "If serv of an obj is in a view, then the object
 is in view" :DEPENDENCY T)
 (INSTANCE ?SERV IS OM-SERVICES
  WITH OBJECT ?OBJ
  WITH APPLN ?APL)
 (AND
  (INSTANCE ?OBJ IS OM-OBJECT WITH-UNKNOWN APPLN ?APL))
THEN
(INSTANCE ?OBJ IS OM-OBJECT
 WITH APPLN ?APL))

(DEFINE-RULE APL-OBJ2
 ()
 (INSTANCE ?APL IS APPLICATIONS
  WITH OBJECTS ?OBJ)
 (AND
  (INSTANCE ?OBJ IS OM-OBJECT WITH-UNKNOWN APPLN ?APL))
THEN
(SLOT-RETRACT-VALUE ?APL OBJECTS ?OBJ))

(DEFINE-RULE APL-OBJ3
 (:DEPENDENCY T)
 (INSTANCE ?OBJ IS OM-OBJECT
  WITH APPLN ?APL)
 (AND
  (INSTANCE ?APL IS APPLICATIONS WITH-UNKNOWN OBJECTS ?OBJ))
THEN
(INSTANCE ?APL IS APPLICATIONS
 WITH OBJECTS ?OBJ))

(DEFINE-RULE APL-REL1
 (:DEPENDENCY T)
 (INSTANCE ?REL IS OM-RELATIONS
  WITH OBJECTS ?OBJ)
WITH APPLN ?APL)

(AND
  (INSTANCE ?APL IS APPLICATIONS WITH-UNKNOWN RELATIONS ?REL))
  THEN
  (INSTANCE ?APL IS APPLICATIONS WITH RELATIONS ?REL))

(DEFINE-RULE APL-REL2
  ()
  (INSTANCE ?APL IS APPLICATIONS WITH RELATIONS ?REL)
  (AND
   (INSTANCE ?REL IS OM-RELATIONS WITH-UNKNOWN APPLN ?APL))
   THEN
   (SLOT-RETRACT-VALUE ?APL RELATIONS ?REL))

(DEFINE-RULE APL-TASK1
  (:DEPENDENCY T)
  (INSTANCE ?TASK IS TASK-SPECS WITH APPLN ?APL)
  (INSTANCE ?APL IS APPLICATIONS WITH-UNKNOWN TASKS ?TASK)
  THEN
  (INSTANCE ?APL IS APPLICATIONS WITH TASKS ?TASK))

(DEFINE-RULE APL-TASK2
  ()
  (INSTANCE ?APL IS APPLICATIONS WITH TASKS ?TASK)
  (AND
   (INSTANCE ?TASK IS TASK-SPECS WITH-UNKNOWN APPLN ?APL))
   THEN
   (SLOT-RETRACT-VALUE ?APL TASKS ?TASK))

(DEFINE-RULE APPL-COBJ-NIL
  ()
  (INSTANCE ?APL IS APPLICATIONS WITH-UNKNOWN CUR-OBJECT ?)
  THEN
  (SLOT-RETRACT-VALUE ?APL CUR-OBJ-ATTRS)
  (SLOT-RETRACT-VALUE ?APL CUR-OBJ-SERV)
  (SLOT-RETRACT-VALUE ?APL CUR-OBJ-PARENTS)
  (SLOT-RETRACT-VALUE ?APL CUR-OBJ-INST-RELNS)
  (SLOT-RETRACT-VALUE ?APL CUR-OBJ-PART-OF)
  (SLOT-RETRACT-VALUE ?APL PARTS-OF-CUR-OBJ))

(DEFINE-RULE CUR-APLN
  ()
  (INSTANCE CURRENT-APPLN IS D-OBJECT
    WITH SLOT1 ?A)
  THEN
  (CUR-APPLN ?A))
(DEFINE-RULE DEL-INST
(:EXPLANATION-STRING "Deletes the instance if no rule of higher priority retracts :yes from delete slot of d-maint-inst."
:PRIORITY -1000)
(INSTANCE D-MAINT-INST IS D-OBJECT
  WITH DELETE :YES
  WITH CUR-INST ?I)
THEN
(DELETE-INSTANCE ?I)
(EVALUATE
  (NOTIFY
    (STR ?I 'IS 'DELETED)))
(SLOT-RETRACT-VALUE D-MAINT-INST DELETE)
(SLOT-RETRACT-VALUE D-MAINT-INST CUR-INST))

(DEFINE-RULE DEL-INST-CHK-APPLN
(:EXPLANATION-STRING "Do not delete object if any application is using it.")
(INSTANCE D-MAINT-INST IS D-OBJECT
  WITH CUR-INST ?I
  WITH DELETE :YES)
(INSTANCE ?I IS ? WITH APPLN ?A)
THEN
(SLOT-RETRACT-VALUE D-MAINT-INST DELETE)
(EVALUATE
  (NOTIFY
    (STR "Cannot delete; used in APPLICATION" ?A))))

(DEFINE-RULE DEL-OBJ-CHK-GENSPEC
(:EXPLANATION-STRING "Do not delete object if it is parent.")
(INSTANCE D-MAINT-INST IS D-OBJECT
  WITH CUR-INST ?I
  WITH DELETE :YES)
(INSTANCE ?R IS GENSPEC-RELN
  WITH OBJECT1 ?I)
(INSTANCE ?I IS OM-OBJECT)
THEN
(SLOT-RETRACT-VALUE D-MAINT-INST DELETE)
(EVALUATE
  (NOTIFY
    (STR "Cannot delete; " ?I "used in inheritance relation" ?R)))

(DEFINE-RULE INCLUDE-APPL
 (:PRIORITY -1000)
 (INSTANCE ?D IS D-OBJECT}
WITH INCLUDE :YES
WITH CUR-INST ?I)
(INSTANCE CURRENT-APPLN IS D-OBJECT WITH SLOT1 ?A)
(INSTANCE ?I IS ORG-MODEL WITH-UNKNOWN APPLN ?A)
THEN
(INSTANCE ?I IS ORG-MODEL
 WITH APPLN ?A)
(EVALUATE
 (NOTIFY "Application view enhanced.")
 (SLOT-RETRACT-VALUE ?D INCLUDE))

DEFINE-RULE INCLUDE-APPL-FAIL (:PRIORITY -100)
(INSTANCE ?D IS D-OBJECT
 WITH INCLUDE ?D
 WITH CUR-INST ?I)
(INSTANCE CURRENT-APPLN IS D-OBJECT WITH SLOT1 ?A)
(INSTANCE ?I IS ORG-MODEL
 WITH APPLN ?A)
THEN
(INSTANCE ?I IS ORG-MODEL
 WITH APPLN ?A)
(EVALUATE
 (NOTIFY "Application view enhancement:" :RETURN :RETURN
 (STR ?I "is already in the view."))
 (SLOT-RETRACT-VALUE ?D INCLUDE))

DEFINE-RULE OM-NO-KEYWORD (:DEPENDENCY T)
(VERIFY OM)
(INSTANCE ?K IS ORG-MODEL
 WITH-UNKNOWN KEYWORDS ?)
THEN
(OM-AGENDA ORG-MODEL ?K MISSING-KEYWORDS))

DEFINE-RULE OM-NO-TASK (:DEPENDENCY T)
(VERIFY OM)
(INSTANCE ?A IS APPLICATIONS
 UNKNOWN
 (INSTANCE ? IS TASK-SPECS
 WITH APPLN ?A))
THEN
(OM-AGENDA APPLICATIONS ?A NO-TASKS))

DEFINE-RULE OM-OBJ-APPLN (:DEPENDENCY T)
(VERIFY OM)
(INSTANCE ?A IS APPLICATIONS
 WITH OBJECTS ?OBJ)
(INSTANCE ?OBJ IS OM-OBJECT WITH-UNKNOWN APPLN ?A)
THEN
(OM-AGENDA ?OBJ ?A)
(DEFINE-RULE OM-OBJ-NO-ATTR
   (:DEPENDENCY T)
   (VERIFY OM)
   (INSTANCE ?OBJ IS OM-OBJECT)
   (UNKNOWN
    (INSTANCE ?OBJ IS OM-ATTRIBUTES
     WITH OBJECT ?OBJ))
   THEN
   (OM-AGENDA OM-OBJECT ?OBJ HAS-NO-ATTRIBUTES))

(DEFINE-RULE OM-SERVICE-NO-POLICY (:DEPENDENCY T)
   (INSTANCE ?S IS OM-SERVICES
    WITH APPLN ?A
    WITH-UNKNOWN POLICY ?P)
   (VERIFY OM)
   THEN
   (OM-AGENDA OM-SERVICES ?S NOT-LINKED-TO-POLICY))

(DEFINE-RULE POLICY1
   ()
   (INSTANCE ?P IS POLICIES
    WITH POLICY-MOD YES)
   (INSTANCE ?OM IS ORG-MODEL
    WITH POLICY ?P
    WITH-UNKNOWN POLICY-MOD ?P)
   THEN
   (INSTANCE ?OM IS ORG-MODEL
    WITH POLICY-MOD ?P))

(DEFINE-RULE POLICY2
   (:DOC-STRING "resets modification indicator after all effects of modifications are identified."
    :PRIORITY -1000)
   (INSTANCE ?P IS POLICIES
    WITH POLICY-MOD YES)
   THEN
   (SLOT-RETRACT-VALUE ?P POLICY-MOD))

(DEFINE-RULE REL-APL1
   (:DEPENDENCY T
    :DIRECTION :BIDIRECTIONAL)
   (INSTANCE ?OBJ1 IS OM-OBJECT
    WITH APPLN ?APL)
   (INSTANCE ?OBJ2 IS OM-OBJECT
    WITH APPLN ?APL)
   (INSTANCE ?REL IS OM-RELATIONS
    WITH OBJECT1 ?OBJ1
    WITH OBJECT2 ?OBJ2)
   THEN
(INSTANCE ?REL IS OM-RELATIONS
  WITH APPLN ?APL))

(DEFINE-RULE REL-APL2
  ()
  (INSTANCE ?REL IS OM-RELATIONS
    WITH OBJECT1 ?OBJ1
    WITH OBJECT2 ?OBJ2
    WITH APPLN ?APL)
  (OR
   (INSTANCE ?OBJ1 IS OM-OBJECT WITH-UNKNOWN APPLN ?APL)
   (INSTANCE ?OBJ2 IS OM-OBJECT WITH-UNKNOWN APPLN ?APL))
  THEN
  (SLOT-RETRACT-VALUE ?REL APPLN ?APL))

(DEFINE-RULE REMOVE-1
  (:EXPLANATION-STRING "Does not allow an object to be dropped
    from a view if its attribute is used in the
    application.")
  (INSTANCE D-MAINT-INST IS D-OBJECT
    WITH CUR-INST ?I
    WITH REMOVE :YES)
  (INSTANCE ?I IS OM-OBJECT)
  (INSTANCE ? IS OM-ATTRIBUTES
    WITH OBJECT ?I
    WITH APPLN ?A)
  (INSTANCE CURRENT-APPLN IS D-OBJECT
    WITH SLOT1 ?A)
  THEN
  (SLOT-RETRACT-VALUE D-MAINT-INST REMOVE)
  (EVALUATE
   (NOTIFY
    (STR "cannot remove object" ?I ", attributes used
    in" ?A))))

(DEFINE-RULE REMOVE-2
  (:EXPLANATION-STRING "Does not allow an object to be dropped
    from a view if its attribute is used in the
    application.")
  (INSTANCE D-MAINT-INST IS D-OBJECT
    WITH CUR-INST ?I
    WITH REMOVE :YES)
  (INSTANCE ?I IS OM-OBJECT)
  (INSTANCE ? IS OM-SERVICES
    WITH OBJECT ?I
    WITH APPLN ?A)
  (INSTANCE CURRENT-APPLN IS D-OBJECT
    WITH SLOT1 ?A)
  THEN
  (SLOT-RETRACT-VALUE D-MAINT-INST REMOVE)
  (EVALUATE
   (NOTIFY
(DEFINE-RULE REMOVE-3
 (:EXPLANATION-STRING "Does not allow an object to be dropped from a view if its attribute is used in the application."
)
(INSTANCE D-MAINT-INST IS D-OBJECT
 WITH CUR-INST ?I
 WITH REMOVE :YES)
(INSTANCE ?I IS OM-OBJECT)
(INSTANCE ? IS GENSPEC-RELN
 WITH OBJECT1 ?I
 WITH OBJECT2 ?02)
(INSTANCE ?02 IS OM-OBJECT
 WITH APPLN ?A)
(INSTANCE CURRENT-APPLN IS D-OBJECT
 WITH SLOT1 ?A)
THEN
(SLOT-RETRACT-VALUE D-MAINT-INST REMOVE)
(EVALUATE
 (NOTIFY
 (STR "cannot remove object" ?I ", child used in" ?A)))
)

(DEFINE-RULE REMOVE-EFFECT
 (:PRIORITY -1000)
 (INSTANCE D-MAINT-INST IS D-OBJECT WITH CUR-INST ?I
 WITH REMOVE :YES)
THEN
(SLOT-RETRACT-VALUE D-MAINT-INST REMOVE)
(EVALUATE
 (SLOT-RETRACT-VALUE ?I
 'APPLN
 (CURRENT-APPLN)))
(EVALUATE
 (NOTIFY
 (STR ?I
 'REMOVED
 'FROM
 'APPLICATION (CURRENT-APPLN))))
)

(DEFINE-RULE SYNONYM1
 ()
 (INSTANCE ?I IS ORG-MODEL)
 (INSTANCE ?OLD IS ORG-MODEL
 WITH SYNONYMS ?I)
THEN
(EVALUATE
 (DELETE-INSTANCE ?I))
(EVALUATE

(NOTIFY
  (STR ?I
    'IS
    'A
    'SYNONYM
    'OF ?OLD)
  )

  :RETURN
  (STR ?I
    'IS
    'RETRACTED)))))

(DEFINE-RULE VIEW-APL-TASK1
 ()
  (VERIFY VIEW)
  (CUR-APPLN ?APL)
  (INSTANCE ?TASK IS TASK-SPECS WITH APPLN ?APL)
  (INSTANCE ?APL IS APPLICATIONS WITH-UNKNOWN TASKS ?TASK)
  THEN
  (INSTANCE ?APL IS APPLICATIONS WITH TASKS ?TASK)
  (AGENDA TASK-SPECS ?TASK INCLUDED-FOR-YOUR-INFO))

(DEFINE-RULE VIEW-APL-TASK2
 ()
  (VERIFY VIEW)
  (CUR-APPLN ?APL)
  (INSTANCE ?APL IS APPLICATIONS WITH TASKS ?TASK)
  (INSTANCE ?TASK IS TASK-SPECS WITH-UNKNOWN APPLN ?APL)
  THEN
  (SLOT-RETRACT-VALUE ?APL TASKS ?TASK)
  (AGENDA ?APL ?TASK RETRACTED-FOR-YOUR-INFO))

(DEFINE-RULE VIEW-APL-TASK3
 ()
  (VERIFY VIEW)
  (CUR-APPLN ?APL)
  (INSTANCE ?APL IS APPLICATIONS WITH TASKS ?TASK)
  (UNKNOWN
   (INSTANCE ?TASK IS TASK-SPECS))
  THEN
  (SLOT-RETRACT-VALUE ?APL TASKS ?TASK)
  (AGENDA ?APL ?TASK RETRACTED-FOR-YOUR-INFO))

(DEFINE-RULE VIEW-NO-KEYWORD
 (:DEPENDENCY T)
 (VERIFY VIEW)
 (CUR-APPLN ?A)
 (INSTANCE ?K IS ORG-MODEL)
WITH-UNKNOWN KEYWORDS ?
   WITH APPLN ?A)
   THEN
   (AGENDA ORG-MODEL ?K MISSING-KEYWORDS))

(DEFINE-RULE VIEW-NO-TASK
   (:DEPENDENCY T)
   (VERIFY VIEW)
   (CUR-APPLN ?A)
   (UNKNOWN
     (INSTANCE ? IS TASK-SPECS
      WITH APPLN ?A))
   THEN
   (AGENDA APPLICATIONS ?A NO-TASKS))

(DEFINE-RULE VIEW-OBJ-APPLN (:DEPENDENCY T)
   (VERIFY VIEW)
   (CUR-APPLN ?A)
   (INSTANCE ?OBJ IS OM-OBJECT WITH-UNKNOWN APPLN ?A)
   (INSTANCE ?A IS APPLICATIONS
      WITH OBJECTS ?OBJ)
   THEN
   (AGENDA OM-OBJECT ?OBJ NOT-IN-VIEW VIEW-UPDATED)
   (SLOT-RETRACT-VALUE ?A OBJECTS ?OBJ))

(DEFINE-RULE VIEW-OBJ-NO-ATTR
   (:DEPENDENCY T)
   (VERIFY VIEW)
   (CUR-APPLN ?A)
   (INSTANCE ?OBJ IS OM-OBJECT
      WITH APPLN ?A)
   (UNKNOWN
     (INSTANCE ? IS OM-ATTRIBUTES
      WITH OBJECT ?OBJ
      WITH APPLN ?A))
   THEN
   (AGENDA OM-OBJECT ?OBJ HAS-NO-ATTRIBUTES))

(DEFINE-RULE VIEW-SERVICE-NO-POLICY (:DEPENDENCY T)
   (VERIFY VIEW)
   (CUR-APPLN ?A)
   (INSTANCE ?S IS OM-SERVICES
      WITH APPLN ?A
      WITH-UNKNOWN POLICY ?)
   THEN
   (AGENDA OM-SERVICES ?S NOT-LINKED-TO-POLICY))
LIST OF REFERENCES


<table>
<thead>
<tr>
<th>Reference</th>
<th>Author(s)</th>
<th>Title</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zmud84</td>
<td>Zmud, R.</td>
<td>&quot;Large Scale Interconnected Information Systems&quot;, Large Scale Systems, 7, 1984, pp 139-149</td>
<td></td>
</tr>
</tbody>
</table>